WebSphere Event Broker

Publish/Subscribe

Version 6 Release 0

WebSphere Event Broker

Publish/Subscribe

Version 6 Release 0

Note
Before using this information and the product it supports, read the information in the Notices appendix.

First Edition (September 2005)

This edition applies to IBM WebSphere Event Broker Version 6.0 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this topic collection

This PDF has been created from the WebSphere Event Broker Version 6.0 GA (September 2005) information center topics. Always refer to the WebSphere Event Broker online information center to access the most current information. The information center is periodically updated on the document update site and this PDF and others that you can download from that Web site might not contain the most current information.

The topic content included in the PDF does not include the ″Related Links″ sections provided in the online topics. Links within the topic content itself are included, but are active only if they link to another topic in the same PDF collection. Links to topics outside this topic collection are also shown, but these attempt to link to a PDF that is called after the topic identifier (for example, ac12340_.pdf) and therefore fail. Use the online information to navigate freely between topics.

Feedback: do not provide feedback on this PDF. Refer to the online information to ensure that you have access to the most current information, and use the Feedback link that appears at the end of each topic to report any errors or suggestions for improvement. Using the Feedback link provides precise information about the location of your comment.

The content of these topics is created for viewing online; you might find that the formatting and presentation of some figures, tables, examples, and so on are not optimized for the printed page. Text highlighting might also have a different appearance.

Part 1. Configuring a publish/subscribe topology

Configuring a publish/subscribe topology

To configure a publish/subscribe topology:

1. Design and configure your broker domain.

   For further information refer to Designing a broker domain and Configuring broker domain components
2. Define the topic trees that you require.

   For further information refer to "Topics" on page 25 and "Adding a new topic" on page 18.
3. Decide which security options to use.

   For further information refer to Publish/subscribe security and Securing the publish/subscribe domain.

Setting up the broker domain for publish/subscribe

Refer to the topics listed below.

Publish/subscribe topologies

A publish/subscribe topology is the brokers, the collectives, and the connections between them, that support publish/subscribe applications in the broker domain.

A publish/subscribe application can consist of a network of brokers connected together. The brokers can all be on the same physical system, or they can be distributed over several physical systems. By connecting brokers together, publications can be received by a client on any broker in the network.

This provides the following benefits:

- Client applications can communicate with a nearby broker rather than a distant one, thereby getting better response times.
- By using more than one broker, more subscribers can be supported.

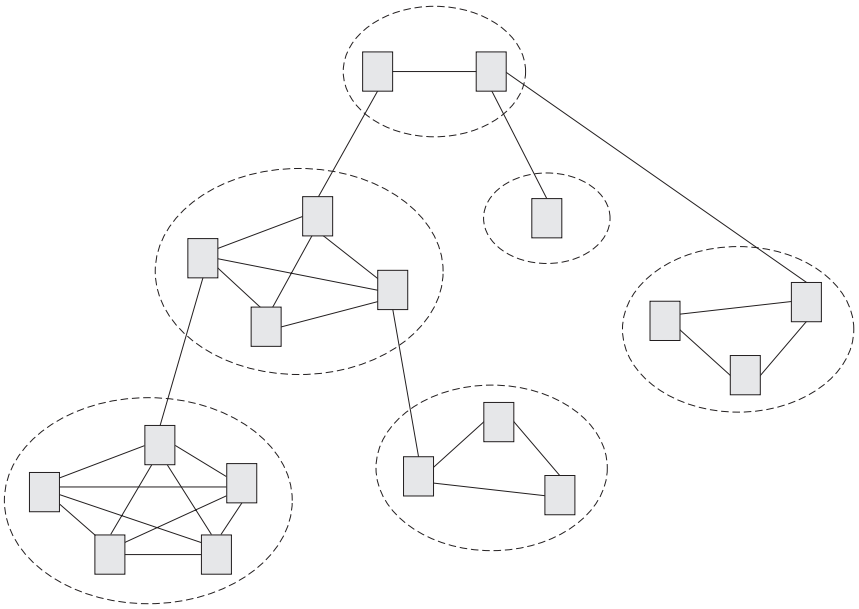
Publications are sent only to brokers that have subscribers that have expressed an interest in the topics being published. This helps to optimize network traffic.

Broker networks

There are three ways of connecting brokers together to make a broker domain:

- Brokers can be simply joined together.
- Brokers can be grouped together into collectives, where a collective is a set of one or more brokers that are directly connected to each other.
- Collectives can be joined together; this is a combination of the previous two ways of grouping brokers together.

The following diagram shows a network of six collectives.



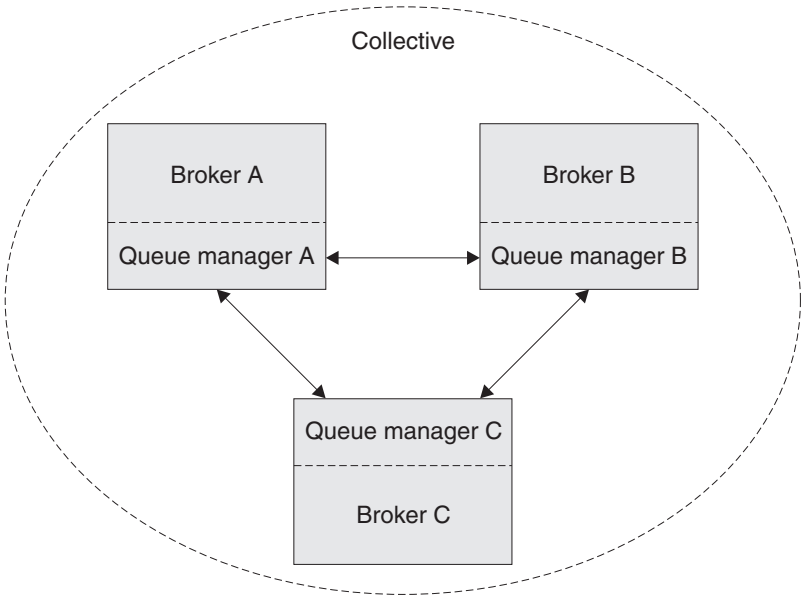
Collectives

A collective is a set of brokers that are fully interconnected and form part of a multi-broker network for publish/subscribe applications.

A broker cannot belong to more than one collective. Brokers within the same collective can exist on physically separate computers. However, a collective cannot span more than one broker domain.

Each pair of broker queue managers must be connected together by a pair of WebSphere MQ channels.

The following figure shows a simple collective of three brokers:



A collective provides the following benefits:

- Messages destined for a specific broker in the same collective are transported directly to that broker and do not need to pass through an intermediate broker. This improves broker performance and optimizes inter-broker publish/subscribe traffic, in comparison with a hierarchical tree configuration.
- If your clients are geographically dispersed, you can set up a collective in each location, and connect the collectives (by joining a single broker in each collective) to optimize the flow of publications and subscription registrations through the network.
- You can group clients according to the shared topics that they publish and to which they subscribe.

  Clients that share common topics can connect to brokers within a collective. The common publications are transported efficiently within the collective, because they pass through only brokers that have at least one client with an interest in those common topics.
- A client can connect to its nearest broker, to improve its own performance. The broker receives all messages that match the subscription registration of the client from all brokers within the collective.

  The performance of a client application is also improved for other services that are requested from this broker, or from this broker's queue manager. A client application can use both publish/subscribe and point-to-point messaging.
- The number of clients per broker can be reduced by adding more brokers to the collective to share workload within that collective.

When you create a collective, the workbench ensures that the connections that you make to other collectives and brokers are valid. You are prevented from making connections that would cause messages to cycle forever within the network. You are also prevented from creating a collective of brokers that does not have the required WebSphere MQ connections already defined.

The queue manager of each broker in a collective must connect to every other queue manager in the collective by a pair of WebSphere MQ channels.

Each broker in the collective maintains a list of its neighbors.

A neighbor can be one of the following:

- a broker in the same collective
- a broker outside its collective to which it has an explicit connection; that is, for which it is acting as a gateway

The complete list of neighboring brokers forms a broker's neighborhood.

Multicast publish/subscribe

In a publish/subscribe system there are client applications, some of which are publishers and some of which are subscribers, that are connected to a network of message brokers that receive publications on a number of topics, and send the publications on to the subscribers for those topics.

Normally, a separate message is sent to each subscriber of a publication. However, with multicast, regardless of how many subscribers to a topic there are on a subnet, only one message is sent. This improves network utilization.

The more subscribers there are in your publish/subscribe system, the greater the improvement to network utilization there might be if you use multicast.

The subscriber must be a JMS client if you want to use Multicast publish/subscribe.

To use multicast, you must change some of the properties of the broker. Some of these properties can be defined as applying to specific topics, but some properties apply to all Multicast messages controlled by that broker.

For each topic you can define whether the topic can be multicast, and the IP address to which Multicast messages are sent.

You can also change those properties in the broker that define, for example, the following:

- The multicast protocol type
- The port that is used for Multicast messages
- A 'Time To Live (TTL)' setting that determines how far from its source a Multicast packet can be sent
- The size of a Multicast packet
- Whether there is a maximum transmission rate and, if there is, its value
- What interface to use for Multicast transmissions

These properties apply to all Multicast messages.

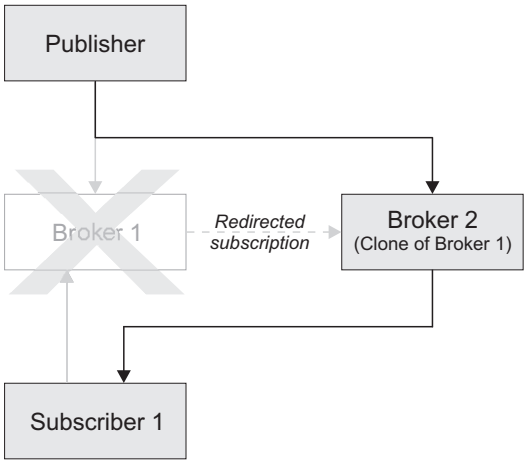
Cloned brokers

A cloned broker is a broker for which you have defined one or more clones; the subscription table of a cloned broker is replicated to all other brokers with which it is cloned.

When a subscriber requests a subscription from a cloned broker, the subscription is also sent to each of the clones of that broker.

Use cloned brokers to improve the availability of your publish/subscribe system. By defining cloned brokers on different computers, you make sure that a publication is delivered to a subscriber even when one of the computers is unavailable.

The diagram shows what happens when Subscriber 1 sends a subscription to Broker 1, but Broker 1 becomes unavailable; because Broker 1 and Broker 2 have been defined as clones, the subscription is redirected to Broker 2 and Subscriber 1 gets the publication from Broker 2.



If two brokers are clones within a collective, duplicate messages might be sent to subscribers registered with brokers inside that collective.

mqsichangeproperties command: The following parameters on the mqsichangeproperties command support cloned brokers:

clonedPubSubBrokerList

This is a set of two pairs. The first item in each pair is the name of a broker, and the second item in each pair is the name of that broker's queue manager.

- Value type — String
- Initial value — null

Migrated topologies

If you have a WebSphere MQ Publish/Subscribe broker network, you can continue to use this network unchanged. The introduction of WebSphere Event Broker to your environment, and the creation of brokers in that broker domain, does not affect your WebSphere MQ Publish/Subscribe broker domain until you take specific action to connect the two networks.

If you want to have two separate, independent networks, you do not have to take any specific actions. You can retain your existing WebSphere MQ Publish/Subscribe network, and install and configure a WebSphere Event Broker network, without any interaction.

Heterogeneous networks: A heterogeneous network is a network of brokers, some of which form a WebSphere MQ Publish/Subscribe network and some of which belong to the WebSphere Event Broker product.

With the WebSphere Event Broker product, there are two ways in which a broker can be joined to the WebSphere MQ Publish/Subscribe network; it can be joined either as a leaf node, or as a parent node.

Leaf node: When a broker is joined as a leaf node, it is joined as a child broker of another broker in the WebSphere MQ Publish/Subscribe network.

Adding the broker as a leaf node rather than as a parent node causes the new broker to receive only some of the WebSphere MQ Publish/Subscribe message traffic that is directed to the brokers for which this new broker is a child broker.

Parent node: When a broker is joined as a parent node, it is joined as a parent broker of one or more brokers in the WebSphere MQ Publish/Subscribe network.

Adding the broker as a parent node rather than as a leaf node causes the new broker to receive all the WebSphere MQ Publish/Subscribe message traffic that is directed to the child brokers for which this new broker is the parent broker.

Changing Broker Topology editor properties

After you have launched the Broker Topology editor in the editor area, you can change or remove the default background image displayed in the editor area.

The following steps show you how to change properties.

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain to display its contents.
3. Double-click **Broker Topology** to launch the Broker Topology editor.
4. Right-click the editor, then click **Properties** to display the Broker Topology editor properties.
5. On the **Editor** page, you can change the background image file, and modify its scale factor in a range of 1 to 5. The default value is 3. Alternatively, you can choose not to display a background image.
6. Optional: On the Description page, provide a description for the background image file.
7. Click **OK** to save your changes and close the Properties dialog.

Any changes you made to the background image are displayed when the Properties dialog closes.

Connecting brokers in a collective

A collective is a set of brokers that are fully interconnected and form part of a multi-broker network for publish/subscribe applications.

You connect brokers in a collective by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.CollectiveProxy.

To connect brokers in a collective:

1. Define the WebSphere MQ channels between the queue managers of each pair of the brokers in the collective; use the standard WebSphere MQ facilities (for example, WebSphere MQ Explorer).
2. Assign the brokers as members of the collective using the Broker Topology editor in the workbench; they do not have to be connected together using the connect function.

Tip: Compare the use of collectives with the use of WebSphere MQ cluster queues, as described in Developing applications using the CMP.

Deleting a collective

You can delete a collective by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.CollectiveProxy.

To delete a collective:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double click the Topology item to open the Broker Topology editor.
4. Right click the collective to be deleted and select Delete, or select the collective to be deleted and click the Delete key, or select Delete from Edit menu.

   The collective is deleted locally, but the delete operation is not completed until you save or close the editor.

Connecting a broker to a collective

You can connect a broker to a collective by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.CollectiveProxy.

To connect a broker to a collective:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double click the Topology item to open the Broker Topology editor.
4. In the Broker Topology editor, click on Connection tool.
5. Click the broker to be connected and then click the collective that you want to connect the broker to.

   The connection is added locally, but the connection is only effective after you have saved, or closed the editor.

Removing a broker from a collective

You can remove a broker from a collective by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.CollectiveProxy.

The following steps show you how to remove a broker from a collective.

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double click the Topology item to open the Broker Topology Editor.
4. Right click the connection that you want to delete and select Delete.

Setting up a multicast broker

You can set up a multicast broker by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.BrokerProxy.MulticastParameterSet.

To make a broker capable of handling multicast requests, do the following tasks from within the workbench:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double-click on the Topology item to open the Broker Topology editor.
4. In the Broker Topology editor, right-click the broker that you want to modify, and select **Properties**.
5. In the left panel of the properties window, select **Multicast**.
6. Select the **Multicast Enabled** check box.
7. Optional: Modify the properties that are listed below; any properties that are not modified take the default value.

   Protocol Type
   The multicast protocol type.

   Valid values are ″PTL″, ″PGM/IP″, and ″UDP encapsulated PGM″. The default value is ″PTL″.

   See “Multicast protocol types” on page 15 for an explanation of these multicast protocol types.

   Min Address
   The lowest IP address that the broker can use for its multicast transmissions.

   This must be in the range 224.0.0.0 through 239.255.255.255. The default value is 224.0.0.0.

   Max Address
   The highest IP address that the broker can use for its multicast transmissions.

   This must be in the range 224.0.0.0 through 239.255.255.255, and must not be lower than the value of Min Address. The default value is 239.255.255.255.

   Data Port
   The UDP data port through which multicast packets are sent and received.

   The default value is 34343.

   Broker Packet Size
   The size, in bytes, of multicast packets.

   This can be in the range 500 through 32000. The default value is 7000.

   Broker Heartbeat Timeout
   The broker sends a control packet periodically, approximately every second, to each client. This packet is used to send various control information, and to keep the heartbeat. The heartbeat timeout value is made known to the clients to help the clients detect a transmitter or

network failure. If a control packet does not arrive within a number, defined as twice the value specified by this parameter, of seconds of the previous control packet's arrival, a client can suspect that there has been a transmitter failure or a network failure.

The default value is 20.

Broker Multicast TTL
The maximum number of hops that a multicast packet can make between the client and the broker. This value is one more than the maximum number of routers that there can be between the client and the broker.

The default value is 1, which means that the multicast packet must remain local to its originator and does not pass through any routers. The maximum value is 255.

Broker Network Interface
The name of the network interface over which multicast packets are transmitted. This is only relevant when the broker is running on a host with more than one network interface.

This can be a host name or an IP address. The default is 'None'. If the default value is chosen, the network interface that is used is operating system dependent.

Overlapping Multicast Topic Behavior
Choose Accept, Reject, or Revert.

The Overlapping Multicast Topic Behavior property controls the behavior of the broker when a client requests a multicast subscription for a topic that is part of a topic hierarchy containing topics that are explicitly disabled for multicast.

For example, consider a topic hierarchy where multicast is a topic with two children, *foo* that is enabled for multicast, and *bar* that is not enabled for multicast.

The three possible settings are:

Accept
A matching multicast subscription is accepted and all publications matching the topic, except those that are specifically excluded, are multicast. In the example shown above, a multicast subscription to *multicast/#* receives messages published on *foo* over multicast, but does not receive any messages published on *bar*.

Reject A multicast subscription to a topic with children that are disabled for Multicast is rejected by the broker. Subscriptions to *multicast/#* are rejected.

Revert Subscriptions to a topic that is disabled for multicast, or has children that are disabled for multicast, result in unicast transmission. A multicast subscription to *multicast/#* receives messages published on *foo* and *bar*, but the messages are sent unicast rather than multicast.

The default value is *Accept*.

Maximum Key Age
The maximum age, in minutes, of a topic encryption key before it must be redefined.

The default value is 360.

8. Optional: Click the + next to Multicast and click **Advanced**. You can now modify the following additional parameters:

Broker Transmission Rate Limit Activation
Use the Broker Transmission Rate Limit Activation property in conjunction with Broker Transmission Rate Limit Value to control network congestion. Choose one of the following values from the drop-down menu:

Disabled
Multicast data is transmitted as fast as possible. If the rate at which messages are submitted to be multicast exceeds the machine or network limits (that is, the speed of Ethernet or the host CPU becomes the bottleneck), these limits define the maximum transmission rate, and message submissions are stopped until all previously submitted messages have been sent.

Static The transmission rate is limited by the value that is specified in Broker Transmission Rate Limit Value.

Dynamic
The limit on the transmission rate can vary during run time, depending on congestion conditions and data losses reported by clients. But the rate never exceeds the Broker Transmission Rate Limit Value.

The default is *Disabled*. If you choose *Static*, you can also choose a value for the parameter Broker Transmission Rate Limit Value.

Broker Transmission Rate Limit Value
This limits the overall transmission rate, in kilobits per second, of multicast packets. This parameter is effective only if the Broker Transmission Rate Limit Activation property is *Static*. This parameter must not exceed the capabilities of the machine or network.

This value can be in the range 10 through 1,000,000.

Client NACK Back Off Time
The maximum time, in milliseconds, that a client listens for another's NACKs before sending its own NACK.

This value can be in the range 0 through 1000. The default value is 100.

Client NACK Check Period
The time, in milliseconds, between periodic checks of reception status and sequence gap detection for NACK building.

This value can be in the range 10 through 1000. The default value is 300.

Client Packet Buffer Number
The number of memory buffers that are created at startup for packet reception. Having a high number of buffers available improves the reception performance and minimizes packet loss at high delivery

rates, but requires increased memory use. Each buffer is 33 KB; having 500 buffers (the default value) uses approximately 15 MB of main memory.

If memory use is important, try using different values for this parameter and look at the effect on the overall performance of your application when transmission rates are high.

This value can be in the range 1 through 5000. The default value is 500.

Client Socket Buffer Size
The size, in kilobytes, of the client's socket receiver buffer. Increasing this value reduces the number of data packets that might be dropped by the client receiver.

This value can be in the range 65 through 10000. The default value is 3000.

Broker History Cleaning Time
The time, in seconds, that is defined for cleaning the retransmission buffer.

This value can be in the range 1 through 20. The default value is 7.

Note: This property is not used in Version 6.

Broker Minimal History Size
The minimum size, in kilobytes, of a buffer that is allocated as an archive for all transmitted packets. This buffer is shared by all reliable topics, and can be used to recover lost packets.

This value can be in the range 1000 through 1,000,000. The default value is 60,000.

Broker NACK Accumulation Time
The time, in milliseconds, that NACKs are aggregated in the broker before recovered packets are sent.

This value can be in the range 50 through 1000. The default value is 500.

Maximum Client Memory Size
The maximum amount of memory, in kilobytes, that can be used by reception buffers in the client.

This parameter is applicable only to PGM multicast protocols. The default value is 262,144 which represents 256 MB.

9. Click OK.
10. Restart the broker; you must do this for the changes that you have made to take affect.

Before you can use multicast, you must define some topics as capable of being multicast.

The recommended way of changing the broker's multicast configuration is to use the workbench. However, you can also use the command **mqsichangeproperties** to change the broker's properties.

The following table relates the properties described above to the corresponding names of the parameters on the **mqsichangeproperties** command that support

multicast. Full details of the **mqsichangeproperties** command is in mqsichangeproperties command.

Property name	mqsichangeproperties parameter
Multicast Enabled	multicastEnabled
Protocol Type	multicastProtocolType
Min Address	multicastAddressRangeMin
Max Address	multicastAddressRangeMax
Data Port	multicastDataPort
Broker Packet Size	multicastPacketSizeBytes
Broker Heartbeat Timeout	multicastHeartbeatTimeoutSec
Broker Multicast TTL	multicastMCastSocketTTL
Broker Network Interface	multicastMulticastInterface
Overlapping Multicast Topic Behavior	multicastOverlappingTopicBehavior
Maximum Key Age	multicastMaxKeyAge
Broker Transmission Rate Limit Activation	multicastLimitTransRate
Broker Transmission Rate Limit Value	multicastTransRateLimitKbps
Client NACK Back Off Time	multicastBackoffTimeMillis
Client NACK Check Period	multicastNackCheckPeriodMillis
Client Packet Buffer Number	multicastPacketBuffers
Client Socket Buffer Size	multicastSocketBufferSizeKbytes
Broker History Cleaning Time (deprecated in V6)	N/A
Broker Minimal History Size	multicastMinimalHistoryKBytes
Broker NACK Accumulation Time	multicastNackAccumulationTimeMillis
Maximum Client Memory Size,	multicastMaxMemoryAllowedKBytes

To enable multicast for the broker WBRK_BROKER use the following command:

```
mqsichangeproperties WBRK_BROKER -o DynamicSubscriptionEngine -n multicastEnabled -v true
```

This enables the broker for multicast, but does not change any other properties of the broker.

To enable multicast for the broker WBRK_BROKER, and to restrict the transmission rate to 50,000 kilobits per second, use the following command:

```
mqsichangeproperties WBRK_BROKER -o DynamicSubscriptionEngine -n multicastEnabled,
 multicastLimitTransRate,multicastTransRateLimitKbps -v true,Static,50000
```

None of the other properties of the broker are changed.

Note the use of commas to separate the properties that are being changed, and also their values.

For the changes to be effective, you must restart the broker.

Warning: Any changes to the broker configuration made using **mqsichangeproperties** are overwritten with the configuration that is held in the Configuration Manager whenever the broker configuration is deployed.

Multicast protocol types

IBM® WebSphere® Event Broker supports three different types of multicast protocol. These are:

- PTL (Packet Transfer Layer)
- PGM/IP
- PGM UDP encapsulated

PTL provides backward compatibility with Version 5, where it is the only multicast protocol that is supported. But for new multicast deployments, either of the two PGM multicast protocols is recommended.

The broker supports two implementations of the PGM multicast protocol, PGM/IP and PGM UDP encapsulated. Which of the two PGM protocol types that you should choose, depends on the complexity of your network topology.

If your network topology consists of two or more subnets with many receiver clients in each subnet, use PGM/IP. PGM/IP takes advantage of PGM router assist support.

For a simpler network topology, use the PGM UDP encapsulated implementation which does not use PGM router assist.

Important: To use PGM/IP, both the broker and the client applications must run with superuser authority. Because of the security risks that are associated with running with superuser authority, it is recommended that no other work should be run on the broker.

Making topics multicast

To make individual topics, or groups of topics, capable of being multicast you need to make changes to the topic hierarchy:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double-click on the Topics item to open the Topics Hierarchy editor.
4. In the Topics Hierarchy editor, right-click the topic, or group of topics, that you want to make capable of being multicast, and select properties.
5. In the left panel of the properties window, select **Multicast**.
6. Choose the **Multicast Enabled** required.

   For the topic root, the choice is either **Enabled** or **Disabled**. The default is **Disabled**.

   For a child topic root, the choice can be **Inherit**, **Enabled**, or **Disabled**. The default is **Inherit**.
7. Check the **Automatic Multicast Address** box, or type in the name of the **MC Group Address**.
8. Choose the **Quality of Service** required. The choice is between **Reliable** or **Unreliable**. The default is **Reliable**.
9. Optional: Select the **Encrypted** check box.
10. Click OK.

Handling high-volume publish/subscribe activity on z/OS

Brokers that handle large numbers of retained subscriptions or publications can use up all the IRLM storage that is allocated by default for DB2 locks. This might cause problems when you try to restart the broker.

The following actions might help stop this happening.

1. Tune the publish/subscribe topology:
   a. Balance execution groups across more brokers; this means that fewer execution groups need to start at the same time and have concurrent locks for the same DB2 subsystem.
   b. Put the brokers in publish/subscribe collectives; this reduces the number of subscriptions in a single broker table and reduces the amount of concurrent access to DB2. See "Publish/subscribe topologies" on page 3 for more information about this.
2. Increase the IRLM storage that is available:
   a. Set the value of MAXCSA so high that the ECSA that is required by the IRLM never reaches this value. Because IRLM gets storage only when it needs it, choose a value that is higher than you expect IRLM to need.
   b. If you are unable to choose a value of MAXCSA sufficiently high that it cannot be exceeded by the ECSA that is required by the IRLM, use the option `PC=YES` on the START irlmproc command. This causes the IRLM to place in its private address space the control block structures that relate to locking. There is more information about this in the DB2 Redbook *DB2 UDB for OS/390 Version 7 Performance Topics, SG24-5351.*

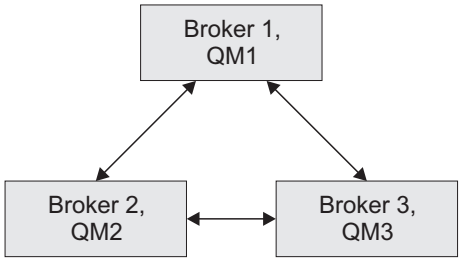
Note: There might be a slight (approximately 1 to 2 percent) performance degradation when you run with `PC=YES`. See *DB2 Universal Database for OS/390 and z/OS Version 7 Administration Guide, SC26-9931* for more information.

Setting up cloned brokers

Each broker that is to be cloned with other brokers must be told which brokers are to be its clones.

To set up three brokers (broker1 with queue manager QM1, broker2 with queue manager QM2, and broker3 with queue manager QM3) to be clones of each other, as shown in the diagram below, use the mqsichangeproperties command for each of the brokers:

1. `mqsichangeproperties broker1 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"broker2,QM2,broker3,QM3\"`
2. `mqsichangeproperties broker2 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"broker1,QM1,broker3,QM3\"`
3. `mqsichangeproperties broker3 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"broker1,QM1,broker2,QM2\"`



Adding a cloned broker

To add a broker to a set of cloned brokers, use the **mqsichangeproperties** command to define the brokers that are its clones, and to tell each of the other brokers that it has a new clone.

To add broker4 (with queue manager QM4) to a set of three cloned brokers (broker1 with queue manager QM1, broker2 with queue manager QM2, and broker3 with queue manager QM3), use the following **mqsichangeproperties** commands:

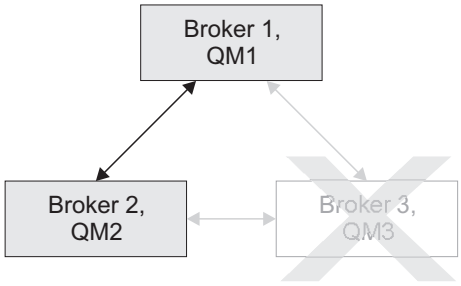
1. `mqsichangeproperties broker4 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"broker1,QM1,broker2,QM2,broker3,QM3\"`
2. `mqsichangeproperties broker1 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"+broker4,QM4\"`
3. `mqsichangeproperties broker2 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"+broker4,QM4\"`
4. `mqsichangeproperties broker3 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"+broker4,QM4\"`

Deleting a cloned broker

To delete a broker from a set of cloned brokers, use the **mqsichangeproperties** command to delete the brokers that were its clones, and to tell each of the other brokers that one of its clones has been deleted.

To delete broker3 from a set of three cloned brokers (broker1 with queue manager QM1, broker2 with queue manager QM2, and broker3 with queue manager QM3), as shown in the diagram below, use the following **mqsichangeproperties** commands:

1. `mqsichangeproperties broker1 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"-broker3\"`
2. `mqsichangeproperties broker2 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"-broker3\"`
3. `mqsichangeproperties broker3 -e default -o DynamicSubscriptionEngine -nclonedPubSubBrokerList -v \"\"`



Operating a publish/subscribe domain

After you have set up your publish/subscribe broker domain, you might want to create or delete topics, or view the current status of your subscriptions.

For information about how to do this, refer to the following topics:

- "Adding a new topic"
- "Deleting a topic" on page 19
- "Querying subscriptions" on page 19

Adding a new topic

You can define a new topic explicitly by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.TopicProxy.

You can define a new topic implicitly by sending to the message broker a Publish command that specifies the new topic. However, to define a new topic explicitly:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double-click on the Topics item to open the Topics Hierarchy editor.
4. Right-click **Topics** in the topics hierarchy displayed by the Topics Hierarchy Editor.
5. From the menu shown, click **Create Topic**; a topic window opens that shows the topic hierarchy.
6. In the topic hierarchy, select the topic that you want to be the parent topic of the topic that you are creating. In the lower pane of the topic window, type the name of your new topic.
7. Click **Next**; the next wizard page opens. The pane on the left of this window shows all the principals (groups and users) that are defined.
8. Select the groups and users that you want to relate to your new topic and click the > icon between the two panes of the window; the pane on the right of the window is updated with the groups and users that you have chosen.
9. For each principal selected in the right-hand pane, you can set **Publish**, **Subscribe**, and **Persistent** attributes by choosing a value from the corresponding list.

By selecting more than one principal, you can choose values for a set of principals.

10. Click **Finish** to insert the topic into the topic hierarchy and update the access control list (ACL) for the topic. The ACL is in a table with four columns that are entitled Principal, Publish, Subscribe, and Persistent. The rows of the table show the properties of each principal that is relevant to the topic.

    The topic is created locally, but the change is not effective until you have saved or closed the editor.

    When saving or closing the editor, you might be prompted to deploy the new topics hierarchy or the deployment might be automatic, depending on the **Perform topics deploy after change** preference.

Deleting a topic

You can delete a topic by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.TopicProxy.

To delete a topic:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double click **Topics** to open the Topics Hierarchy Editor.
4. In the Topics Hierarchy editor, right-click the topic that you want to delete, and select **Delete**; alternatively, select the topic that you want to delete and press the Delete key, or select Delete from the Edit menu.

   The topic is deleted locally, but the delete is not effective until you do a save.

Querying subscriptions

You can query a topic by either using the Message Brokers Toolkit or by using the Configuration Manager Proxy Java API. This topic describes how to use the Message Brokers Toolkit. For information about how to use the Configuration Manager Proxy (CMP), see Developing applications using the CMP and Class com.ibm.broker.config.proxy.TopicProxy.

To query a subscription:

1. Switch to the Broker Administration perspective.
2. In the **Domains** view, click **Subscriptions** from the list of domain objects shown; the Subscriptions Query Editor opens in the workbench.

   You can also open the editor by double-clicking the **Subscriptions** item in the tree, or by right-clicking the **Subscriptions** item and clicking **Open**, or by clicking the **Subscriptions** item and clicking **Enter**.
3. Fill in the fields that are required to generate your subscriptions query.

   To generate your query, you might not need to fill in all the fields shown.
4. Click **Query**. The results of your query are displayed in the lower part of the edit window.

Part 2. Developing publish/subscribe applications

Developing publish/subscribe applications

The following information shows you how publications and subscriptions flow through the network and tells you about different subscribers and publishers.

How publications and subscriptions flow through the network

The transport mechanism that you choose determines how publications and subscriptions flow through a network. The transports that are available are described in End-user application support

MQ Subscribers and Publishers

When a client registers a subscription, the broker registers a matching subscription with its neighbors. This is called a 'proxy subscription'. If an identical subscription has already been registered, the broker does not register again; only one proxy subscription is in effect at any one time. Similarly, when a client deregisters a subscription from a broker, the broker deregisters the proxy subscription from its neighbors, if the client is the only client for which the broker is holding the proxy.

Content-based filters are not included in proxy subscriptions. A super-set of messages might be received by the broker to which a subscriber that specified a content filter is registered, but a message is not passed on to that subscriber by its local broker unless there is a content match.

All proxy subscriptions are made with the PersistenceAsPublisher option. This results in messages being delivered to neighboring brokers with the persistence specified by the publisher. Client subscription persistence options only take effect at the local broker; that is, at the broker with which the clients have registered.

A subscriber that requests persistent delivery always receives a persistent message for matching publications. However, the message might be delivered through the broker network as a nonpersistent message if this was specified by the publisher. If a problem occurs during the transmission of a message between publisher and subscriber, the subscriber might never get the message despite specifying persistent delivery as an option on subscription registration.

Real-time Transport Subscribers and Publishers

When two neighboring brokers contain a message flow that has either a Real-timeInput node or a Real-timeOptimizedFlow node, a connection is made between the two brokers using the broker host and broker port parameters that are configured as part of the broker.

Subscriptions and 'proxy subscriptions' are not forwarded to neighboring brokers for clients that subscribe using Real-time Transport.

Real-time Transport publication messages are forwarded to all neighboring brokers, even if there are no Real-time Transport subscriptions there to match.

"Multicast publish/subscribe" on page 5 can be used to improve network utilization.

Publish/Subscribe

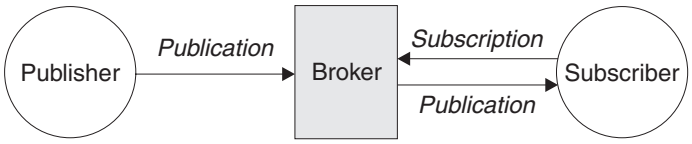
Publish/subscribe is a style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers) using a broker.

In a publish/subscribe system, a publisher does not need to know who uses the information (publication) that it provides, and a subscriber does not need to know who provides the information that it receives as the result of a subscription.

Compare this with a point-to-point style of messaging application, in which the application that sends messages needs to know the destinations of the messages that it sends.

Message brokers make sure that messages arrive at the correct destinations, and are transformed to the format required at each destination.

The following figure shows the simplest publish/subscribe application. There is one publisher, one broker, and one subscriber. A publication is sent from the publisher to the broker, a subscription is sent from the subscriber to the broker, and the publication is then sent from the broker to the subscriber.



However, a typical publish/subscribe system has more than one publisher and more than one subscriber, and often more than one broker. An application can be both a publisher and a subscriber.

The publisher generates a message that it wants to publish and defines the topic of the message. A message flow running in the broker retrieves the message from its input node and passes the message to a Publication node for distribution to all subscribers that have registered an interest in the topic.

The input node might be one of the following built-in nodes:

- An MQInput node which represents a WebSphere MQ queue
- A Real-timeInput node which receives messages from a JMS application using WebSphere MQ Real-time Transport
- SCADAInput which represents a SCADA input port

A subscriber registers a request for a publication by specifying one of the following items:

- The topic, or topics, of the published messages that it is interested in.
- The subscription point from which it wants to receive publications.
- The content filter that should be applied to the published message.
- The name of the queue (known as the subscriber queue) on which publications that match the criteria selected should be placed. This can be the name of a cluster queue so that publications can be distributed to clustered subscribers.

Topics

A topic is a character string that describes the nature of the data that is published in a publish/subscribe system.

Topics are key to the successful delivery of messages in a publish/subscribe system. Instead of including a specific destination address in each message, a publisher assigns a topic to the message. The message broker matches the topic with a list of clients (subscribers) who have subscribed to that topic, and delivers the message to each of those clients.

Note that a publisher can control which subscribers can receive a publication by choosing carefully the topic that is specified in the message.

Topics can be defined by a system administrator using the workbench. However, the topic of a message does not have to be defined before a publisher can use it; a topic can also be defined when it is specified in a publication for the first time.

More than one topic can be specified for a publication.

A topic string can include any character from the Unicode character set, including the space character. However, there are three characters that have special meanings. These characters (″/″, ″#″, and ″+″) are described in “Special characters in topics” on page 45.

Although a null character does not cause an error, do not use null characters in your topic strings.

Topic trees

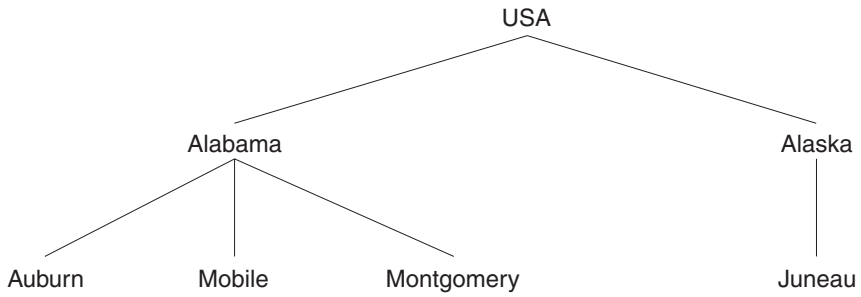
Although you can use any name for a topic, choose a name that fits into a hierarchical tree structure. Thoughtful design of topic names and topic trees can help you with the following operations:

- Subscribing to multiple topics.
- Establishing security policies.
- Automatically reacting to messages on a specific topic; for example, by sending an alert to a manager's pager.

Each topic that you define is an element, or node, in the topic tree. The topic tree can either start empty or contain topics that have been defined by a system administrator using the workbench. You can define a new topic either by using the workbench or by specifying the topic for the first time in a publication.

Although you can construct a topic tree as a flat, linear structure, it is better to build a topic tree in a hierarchical structure with one or more root topics.

The following figure shows an example of a topic tree with one root topic:



Each character string in the figure represents a node in the topic tree. A complete topic name is created by aggregating nodes from one or more levels in the topic tree. Levels are separated by the ″/″ character. The format of a fully specified topic name is: ″root/level2/level3″.

The valid topics in the topic tree shown above are:
″USA″
″USA/Alabama″
″USA/Alaska″
″USA/Alabama/Auburn″
″USA/Alabama/Mobile″
″USA/Alabama/Montgomery″
″USA/Alaska/Juneau″

When you design topic names and topic trees, remember that the message broker does not interpret, or attempt to derive meaning from, the topic name itself. It uses the topic name only to send related messages to clients who have subscribed to that topic.

Publishers

A publisher is an application that makes information about a specified topic available to a broker in a publish/subscribe system.

In a publish/subscribe system, an application, known as the publisher, can send a message to a message queue or port that is associated with an input node in a message flow that contains a Publication node. Depending on the transport used by the publisher, this input node might be an MQInput node, a SCADAInput node, or a Real-timeInput node.

Another application, known as the subscriber, can send a subscription request to the broker, which then sends relevant publication messages to the subscriber's message queue or port.

A published message can be requested by more than one subscriber, and a subscriber can request messages, on the same or different topics, from more than one publisher.

Publications

A publication is a piece of information about a specified topic that is available to a broker in a publish/subscribe system.

Typically, a broker distributes a publication that it receives to all applications that are connected to it and that have registered a subscription for the publication. The

broker also distributes the publication to all other brokers connected to it, either directly or through a network of brokers that have subscribers for the publication.

Local publications

Publishers can restrict access to their publications to only those subscribers that are registered to the same broker as the publisher. This publication is known as a local publication. Local publications are not forwarded to other brokers.

Global publications

A publication whose distribution is not restricted to only those subscribers that are registered to the same broker as the publisher is known as global publication. A global publication is forwarded to all brokers, connected either directly or through a network of brokers, that have one or more subscribers for the publication.

Retained publications

Typically, a broker discards a publication after it has been sent. However, a publisher can specify (in the case of the *Publish* message, by specifying the *RetainPub* option) that it wants the broker to keep a copy of the publication, which is then called a retained publication.

If a retained publication has been published, new subscribers to that publication receive the publication without having to wait for it to be published again.

For example, a subscriber that registers a subscription for a stock price receives the latest published stock price immediately, and does not have to wait for the stock price to be republished.

A broker retains only one publication for each combination of topic and subscription point.

State and event information

Information being published can be categorized either as state information or as event information.

State information is information about the current state of something. The current price of stock or the current score in a soccer match are both examples of state information.

Event information is information about an individual event that occurs. A change in the price of stock or the scoring of a particular goal in a soccer match are both examples of event information.

When an event occurs, the current state information is no longer required and is superseded by new state information.

If a publication contains state information, it is often published as a retained publication. A new subscriber typically wants the current information immediately; the subscriber does not want to wait for an event that causes the information to be republished.

Subscribers

A subscriber is an application that requests information about a specified topic from a publish/subscribe broker.

The subscribing application might be a WebSphere MQ, WebSphere Event Broker, WebSphere MQ Everyplace, SCADA, or JMS/IP application.

The subscriber sends a subscription request to a broker, specifying which publications it wants to receive. The request defines the topic, the filter, and the subscription point of each publication, and also specifies the name of a queue to which the publications should be sent. This queue is known as the subscriber queue.

Messages that are published by a publisher can be received by more than one subscriber, and a subscriber can receive messages, on the same or different topics, from more than one publisher.

Subscriptions

A subscription is a record that contains the information that a subscriber passes to its local broker to describe the publications that it wants to receive.

A subscription consists of the following information:
- One or more topics; wildcard characters can be used.
- An optional subscription point.
- An optional filter on the contents of the publication message.
- A subscriber queue, queue manager, and optional *CorrelId*.

Subscribers issue subscription registration requests to their local broker when they want to receive published messages. All the information associated with the subscription is recorded by the broker in the broker's subscription table.

When the broker receives a publication, it scans its subscription table to determine whether there is a subscription request that matches the topics, subscription point, and filter, of the publication. For each subscription request that matches, the broker forwards the publication to the subscriber queue that is specified, unless the subscriber has requested, by specifying the PubOnReqOnly option on its request, that it only wants publications that are newly published.

A subscription is removed from the subscription table only when one of the following events occurs:
- The subscriber deregisters the subscription.
- The subscription expires in the transports supported.
- The subscriber application ends.
- The subscription is deleted by the system administrator using the workbench.
- A temporary dynamic queue that is specified by the subscriber as the queue to which publications should be sent, is deleted.

Filters

A filter is an expression, which might include wildcard characters, that is applied to the content of a publication message to determine whether it matches a subscription.

When you register a subscription, in addition to specifying a topic and subscription point, you can specify a filter to select publications according to their contents. WebSphere Event Broker needs to know how to parse the contents of the message correctly. This can be achieved in a number of ways:
- The message is a self-defining XML message.
- The message template is defined in the MQRFH2 header.

The filter itself is entered as an SQL-like expression; for example:

```
Body.Name LIKE 'Smit%'
```

This means that the contents of a field called Name in the body of a publication message are extracted and compared with the string given in the expression. If the string in the message starts with the characters ″Smit″, the expression evaluates to TRUE and the publication is sent to the subscriber.

If you want to select publications using filters only, without specifying a topic, you can register a subscription with the required filter and a topic of ″#″ (all topics). You then receive publications on only those topics for which you have access authority.

This subscription results in all publications from all connected brokers being sent to the broker that is local to the subscriber. Therefore, for performance reasons, if you have set up a network of brokers, you are advised to not use this technique.

Subscription points

A subscription point is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

A subscriber that registers a subscription without specifying a subscription point receives publications from any unnamed Publication node in the message flow, provided that there is a match with the topic and filter specified by the subscriber.

This applies to all message flows running in all brokers connected in the same network, unless the local has been specified when registering the subscription.

Using subscription points

If you have more than one Publication node in a message flow, you can differentiate between them by specifying subscription points. Choose values that indicate the type of message that is routed to each Publication node.

Example

Consider an application that publishes stock prices. The prices that are available from the first Publication node in the message flow are in dollars. This Publication node uses the default subscription point.

You can define a second path through the message flow that takes the price in dollars, and converts this using a defined conversion value, to produce the same message but with the stock price in pounds. These messages are published at a second Publication node that has its subscription point property set to ″Pounds″.

You might have another message flow, in the same broker or a connected broker, that publishes stock prices in pounds on the same topic. Make sure that it uses the ″Pounds″ subscription point, and that all other message flows that publish their stock prices in dollars use the default subscription point.

Subscribers specifying the relevant topic (for example, ″stock″) can then choose whether to receive the information in dollars or in pounds, by using the default subscription point or the ″Pounds″ subscription point when they subscribe.

Performance considerations for Real-time transport

A broker that is configured to use Real-time transport has several properties that can be changed to affect the behavior of the broker. These properties are:

brokerInputQueues
This property defines how many queues are available to store incoming messages; the higher the number of queues, the higher the potential rate of accepting incoming messages by the broker.

The default value is 1.

brokerInputQueueLength
This property defines the maximum number of messages that can be stored in each input queue; the higher the value, the higher the number of input messages that can be stored in each input queue; however, be aware that the higher the value of this property, the larger the amount of memory that the broker requires for each queue.

The default value is 99.

maxBrokerQueueSize
This property defines the maximum size of the broker's output queues. If this maximum is exceeded, the broker deletes all messages queued to that broker, except the latest message, any high-priority messages, and any response messages. If this property is set to 0, the broker does not impose any limit on the number of bytes that can be queued to another broker.

The default value is 1000000 bytes.

brokerPingInterval
This property defines the time in milliseconds between broker-initiated ping messages on broker-broker connections. Ping messages are used to confirm that communications are still open between both sides of the connection. If the value is 0, no ping messages are sent by the broker.

The default value is 5000 milliseconds.

maxMessageSize
This property defines the maximum size of message that can be received by the broker. If the broker receives a message that is bigger than this, the broker disconnects the client that sent the message.

The default value is 100000 bytes.

Use the mqsichangeproperties command to define new values for these properties if you don't want to use the default values.

WebSphere MQ Publish/Subscribe

WebSphere MQ Publish/Subscribe provides publish/subscribe application support for WebSphere MQ applications.

Before Fix Pack 8 of WebSphere MQ Version 5.3, this support was only available by using SupportPac MA0C, details of which can be found at the following web site: WebSphere MQ (MQSeries) Publish/Subscribe.

More information about WebSphere MQ publish/subscribe support is given at "WebSphere MQ Publish/Subscribe" on page 52.

Streams

A stream is a method of topic partitioning used by WebSphere MQ Publish/Subscribe applications. Sets of related topics are grouped together into separate streams.

By using streams, different security controls can be applied to different groups of topics, and the publishing workload of the broker can be better balanced.

Although WebSphere Event Broker provides other ways for an application to achieve both of these behaviors, the concept of streams is supported for compatibility with MQRFH applications.

WebSphere Event Broker allows MQRFH client applications to specify an MQPSStreamName command parameter in their subscriptions and publications. However, the stream name is used only to modify the topic to preserve the partitioning characteristic of WebSphere MQ Publish/Subscribe.

If the name of a stream associated with a message is not SYSTEM.BROKER.DEFAULT.STREAM, the message is processed as if the topic, or topics, mentioned within the message had been prefixed with the string "$SYS/STREAM/<streamname>/". For example, a subscription to Topic1 that specifies a stream name of StreamX is processed as if the subscription had been made to topic "$SYS/STREAM/StreamX/Topic1".

MQRFH2 publishing and subscribing applications can also target stream-related topics, even though they themselves cannot specify a stream name in the messages they send to the WebSphere Event Broker broker. To do this, they must prefix the topics with the appropriate stream prefix.

For example, an MQRFH2 subscriber must specify topic "$SYS/STREAM/STOCK.STREAM/IBM/Latest" to subscribe to topic "IBM/Latest" that is published on stream STOCK.STREAM within the WebSphere MQ Publish/Subscribe network.

WebSphere MQ Publish/Subscribe allows a stream-related publication to be sent only to a queue having the same name as the stream. However, WebSphere Event Broker allows publishing clients to send their publications to any input queue in a message flow. MQRFH applications choosing explicitly to specify a stream name parameter within a publication can send it to any publication queue being serviced by the WebSphere Event Broker broker. The name of the queue does not have to be the same as the name of the stream. However, this behavior might affect the order in which publications are received. Consider whether this is important for your applications.

Each Publication node has an *Implicit Stream Naming* property that defaults to *true*. This default option results in behavior that is identical to that in WebSphere MQ Publish/Subscribe when an MQRFH publication does not contain an explicit stream name. If this property is *false*, and the publication contains no explicit stream name, SYSTEM.BROKER.DEFAULT.STREAM is assumed.

The options that are available to both MQRFH and MQRFH2 client applications that publish messages are shown in the following table; the table shows the options for both the default stream and an example stream name of *StreamX*.

	MQRFH publisher		MQRFH2 publisher	
	default stream	StreamX	default stream	StreamX
MQRFH subscriber	S1,P1	S2,P2	S1,P3	S2,P4
MQRFH2 subscriber	S3,P1	S4,P2	S3,P3	S4,P4

Subscriber notes

S1 Subscriber subscribes either without a stream name or with stream name ″SYSTEM.BROKER.DEFAULT.STREAM″.

S2 Subscriber subscribes with stream name ″StreamX″.

S3 Subscriber subscribes on topic without adding ″$SYS/STREAM/<streamname>/″.

S4 Subscriber subscribes prefixes topic with ″$SYS/STREAM/StreamX/″.

Publisher notes

P1 Publisher publishes on any queue specifying stream name ″SYSTEM.BROKER.DEFAULT.STREAM″, or publishes without specifying a stream name on any queue with the *Implicit Stream Naming* property set to false.

P2 Publisher publishes on any queue specifying stream name ″StreamX″, or publishes without specifying a stream name on queue ″StreamX″ with the *Implicit Stream Naming* property set to true.

P3 Publisher publishes on any queue without adding the prefix ″$SYS/STREAM/<Stream>/″ to the topic.

P4 Publisher publishes on any queue and adds the prefix ″$SYS/STREAM/StreamX/″ to the topic.

Note: The ″$SYS/STREAM/<streamname>/″ prefix is removed from all topics in an MQRFH2 publication when it is delivered to an MQRFH subscriber.

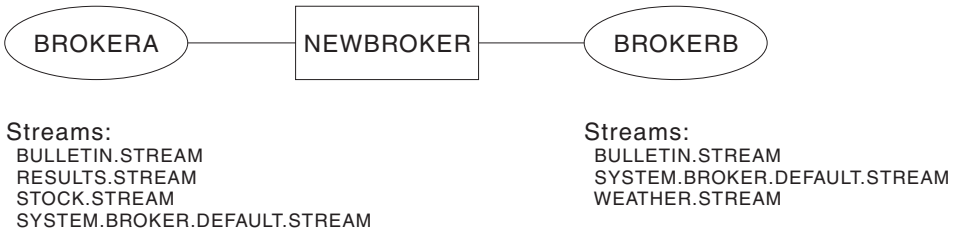
Streams and neighboring brokers: In a WebSphere MQ Publish/Subscribe network, a broker does not have to support the same set of streams as its neighbors. If a broker does not support a stream that is supported by one of its neighboring brokers, publications associated with that stream are not available to clients at that broker.

When an WebSphere Event Broker broker joins the network, it supports all the streams of its neighboring WebSphere MQ Publish/Subscribe brokers. This means that clients of the WebSphere Event Broker broker can target publications for any stream supported by any of its WebSphere MQ Publish/Subscribe neighbors.

However, to make these publications available, you must define the stream queues, and define and deploy the message flows that support them, to the WebSphere Event Broker broker.

The effects of adding an WebSphere Event Broker broker into a multi-stream WebSphere MQ Publish/Subscribe environment are illustrated in the following figure. The WebSphere Event Broker broker, NEWBROKER, has been used to join WebSphere MQ Publish/Subscribe brokers, BROKERA, and BROKERB.

A heterogeneous network



The default stream queue SYSTEM.BROKER.DEFAULT.STREAM is always supported by every broker in an WebSphere MQ Publish/Subscribe network, and must be defined at every WebSphere Event Broker broker in a heterogeneous network. At each broker, you must define and deploy a message flow to service this queue.

When a WebSphere Event Broker broker is integrated into an WebSphere MQ Publish/Subscribe network, and links two or more WebSphere MQ Publish/Subscribe brokers that share common streams, you must define the common stream queues, and define and deploy the message flows that service them, to the WebSphere Event Broker broker.

For example, the WebSphere Event Broker broker NEWBROKER must have a stream queue defined for BULLETIN.STREAM. It must also have a message flow defined and deployed to provide a publication service for that queue.

You need to define stream queues and associated message flows to the WebSphere Event Broker broker for other streams shown in the figure only if one of its WebSphere MQ Publish/Subscribe neighbors can send a message to one of these queues. A message is sent if one of the following events occurs:

1. A subscription to a publication on one of these streams is registered by a client of the WebSphere Event Broker broker.
2. A *DeletePublication* command for the stream is issued by a client anywhere within the broker network.

If you are unsure about whether the above cases might occur, create stream queues and message flows in the WebSphere Event Broker broker for every stream that is supported by an WebSphere MQ Publish/Subscribe neighbor. If you do not do this, the following might happen:

- Messages sent from WebSphere MQ Publish/Subscribe brokers are put on the dead-letter queue (DLQ) of the WebSphere Event Broker broker if the stream queue does not exist on that broker.
- Messages build up on stream queues on the WebSphere Event Broker broker if the stream queue exists but no message flow is deployed to service it.

Streams and migration: When an WebSphere MQ Publish/Subscribe broker is migrated to an WebSphere Event Broker broker (using the **migmqbrk** command), the streams supported at the time of the migration are replicated exactly in the WebSphere Event Broker broker. No changes can be made subsequently; that is, no streams can be added to, or removed from, this replicated set. The migration is not complete until you have created and deployed message flows that process all these streams.

Stream authority

In WebSphere MQ Publish/Subscribe, all publish and subscribe authority checks are performed against the stream queue. Publishing applications need authority to

put messages to the stream queue. The WebSphere MQ Publish/Subscribe broker also checks the authority of subscribing applications which require browse authority on the stream queue. A subscribing application also needs to have put authority for the queue that it nominates to receive its publications.

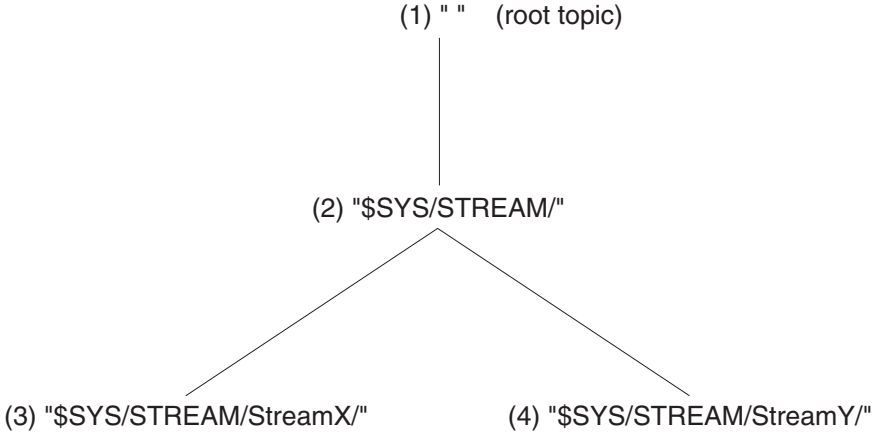
A similar check is made by WebSphere Event Broker brokers, but there is no checking for subscribe, or browse, authority. Instead, WebSphere Event Broker uses Access Control Lists (ACLs), which you can create using the workbench, to provide the required authorities for individual topics. For more information about ACLs, see Security for runtime resources.

Before you migrate an WebSphere MQ Publish/Subscribe broker to WebSphere Event Broker, or migrate your WebSphere MQ Publish/Subscribe applications to run on a WebSphere Event Broker, you must consider the following security implications:

- Publishing applications are subject to the same checks even if your broker is not running with topic security enabled, because the authority to put a message to the stream or publication queue continues to be checked by WebSphere MQ.

  However, stream publications can be processed by WebSphere Event Broker on any input queue, because publishers no longer need to put to a queue with the same name as the stream. Therefore, set up equivalent ACLs for all streams using their corresponding topic level qualifiers
- The WebSphere Event Broker broker does not check that subscribing applications have browse authority on the stream queue. Instead, WebSphere Event Broker models streams by prefixing all topics that aren't part of the default stream with a unique prefix, $SYS/STREAM/<streamname>/. This maintains the partitioning characteristics of streams and also allows stream-specific ACLs to be set up. Because topics in the default stream are not altered by the broker, the root topic can be used to specify authorities for default stream topics.

Stream authorities



The figure shows the stream authorities that are required. This example assumes that you have updated the default ACL on the topic root for principal PublicGroup with authority for publish, subscribe, and persistent delivery all set to `deny`.

Using this example, assume that the following groups are defined:
PDefault: the group of users authorized to publish on the default stream
SDefault: the group of users authorized to subscribe to the default stream
PStreamX: the group of users authorized to publish on StreamX

SStreamX: the group of users authorized to subscribe to StreamX
PStreamY: the group of users authorized to publish on StreamY
SStreamY: the group of users authorized to subscribe to StreamY

You must grant and deny authorities by setting up ACLs as follows:

1. PDefault must be granted publish authority on the root, and SDefault must be granted subscribe authority on the root.
2. PDefault must be denied publish authority on $SYS/STREAM/, and SDefault must be denied subscribe authority on $SYS/STREAM/.

   These settings ensure that publishers and subscribers on the default stream are unable to publish on, or subscribe to, other streams without an explicit ACL that overrides the relevant setting.
3. PStreamX must be granted publish authority on $SYS/STREAM/StreamX/, and SStreamX must be granted subscribe authority on $SYS/STREAM/StreamX/.

   These settings override any setting on parent topics and limit publish and subscribe activity to users within these specific groups.
4. PStreamY must be granted publish authority on $SYS/STREAM/StreamY/, and SStreamY must be granted subscribe authority on $SYS/STREAM/StreamY/.

   These settings override any setting on parent topics and limit publish and subscribe activity to users within these specific groups.

If you want to set up exceptions to this situation, you can do so by introducing an ACL at the appropriate point. For example, if you wanted to grant authority to publishers to the default stream, PDefault, to publish on StreamX, you must create an explicit ACL at point (3) to grant that authority; this overrides the denial of authority at point (2). In this scenario, users in PDefault would still be unable to publish on StreamY.

Publishing

The publisher sends a Publish command message to the input node of a message flow that contains a Publication node. The publisher must have the necessary access authority, set by the WebSphere Event Broker system administrator, to publish on the topic or topics that are specified for this publication.

Using retained publications

By default, a broker discards a publication after it has sent that publication to all interested subscribers. However, a publisher can specify that it wants the broker to keep a copy of a publication, which is then called a retained publication.

A copy of a retained publication is sent by the broker to all subscribers who register an interest in the topic of the publication. This means that a new subscriber does not have to wait for information to be published again before they receive it.

For example, a subscriber that registers a subscription to a stock price receives the most recently published price straightaway, without having to wait for the stock price to change and be republished.

If RetainPub is specified as a publication option in the Publish message, the publication is retained by the broker and replaces any previously retained publication for that topic.

Because the broker retains only one publication for each topic and subscription point, the old publication is deleted when a new publication arrives.

When deciding whether to use retained publications, consider the following questions.

- Do your publications contain state information or event information?

  Typically, event information is not retained, but state information is retained. However, if all the subscriptions to a topic are in place before any publications are made on that topic, and if no new subscriptions are expected, there is no need to retain publications, even for state information, because the publications are delivered to all subscribers as soon as they are published.

  Publications that contain state information might also not need to be retained if they are published very frequently (for example, every second). With this frequency of publishing, any new subscriber (or a subscriber recovering from a failure) receives the current state almost immediately after it subscribes.
- Do you want to receive publications only when you request them?

  If you use retained publications, subscribers can register using the PubOnReqOnly option of the Register Subscriber message. This means that the broker sends publications to a subscriber only when that subscriber requests an update. The broker then sends to the subscriber the current retained publication that matches the subscription.
- Can retained publications be mixed with non-retained publications on the same topic?

  This is not recommended. If you have a retained publication, and then publish a non-retained publication on the same topic, the existing retained publication is still retained. It is not updated by the non-retained publication. If a subscriber registers using the *PubOnReqOnly* option of the Register Subscriber message, it cannot access any non-retained publications. The broker sends only the current retained publication in response to a request for an update.
- Can you have more than one application publishing retained publications on the same topic?

  You are recommended to not have more than one application publishing retained publications on the same topic. If you do, and the timing is close to simultaneous, it is indeterminate which publication is retained. If the publishers use different brokers, different retained publications for the same topic might be held at each broker.
- How does the subscriber application recover from failure?

  If the publisher does not use retained publications, the subscriber application might need to store its current state locally. If the publisher uses retained publications, the subscriber can request an update to refresh its state information after a restart.

  The broker continues to send publications to a registered subscriber even if that subscriber is not running. This can lead to a buildup of messages on the subscriber queue. This buildup can be avoided if the subscriber registers with the *PubOnReqOnly* option of the Register Subscriber message. The subscriber must then refresh its state periodically either by requesting an update or by using a temporary dynamic queue.
- What are the performance implications of retaining publications?

  The broker needs to store retained publications in a database. This reduces throughput. If the publications are very large, a considerable amount of disk

space is needed to store the retained publication of each topic. In a multi-broker environment, retained publications are stored by all other connected brokers that have a matching subscription.

Use the *Expiry* field of the message descriptor (MQMD) to set an expiry interval for a retained publication.

The sample verification applications that are shipped with WebSphere Event Broker include the Soccer Results service. This sample uses retained publications to record the latest score in each soccer match that it is monitoring. The sample code illustrates the programming that is required to support this option.

Not all applications can publish retained publications, and not all retained publications can have expiry dates applied to them. The following table shows which applications can publish retained publications and whether the retained publications can have an expiry date:

	MQ	**SCADA**	**JMS/IP**
Retained	YES	YES	NO
Expiry Date	YES	NO	NO

The columns in the table indicate four types of application. The first row indicates whether a publication can be a retained publication, and the second row indicates whether an expiry date can be applied to the publication.

Subscribing

A subscriber registers a request for a publication by specifying the following elements.

- The topic, or topics, of the published messages that it is interested in.

  Wildcard characters can be used when subscribing to topics, and can be used at any level in the topic name string. By creating your applications so that topics are defined in well-structured topic trees, the applications can subscribe to sub-trees by placing the multilevel wild card ″#″ at the end of a topic.

  Note that, although the single-level wild card is accepted anywhere in the topic name, performance is better when it is placed at the end of the string.

  You can specify more than one wildcard character within a subscription. For example, ″+/Alabama/#″ is a valid topic.

  Note that, if you subscribe with topic ″#″, you receive all publications from all connected brokers. This might result in a very overloaded broker network.
- The subscription point from which it wants to receive publications.

  This value should match the subscription point property that is set for at least one publication node defined in this broker. If the value does not match any existing subscription point, the subscriber does not receive any publications, unless a publication node is defined subsequently with this subscription point name.

  If you do not specify a subscription point, the default description point is assumed. You receive all publications that have matching topics and filters.

  For SCADA applications, the SCADA connection port is the implied subscription point.
- The content filter that should be applied to the published message.

This information is optional. If you do not specify a content filter, all published messages with matching subscription points and topics are received.

Content filters cannot be used with SCADA messages.

- The name of the queue (known as the subscriber queue) on which publications that match the criteria selected should be placed. This queue must exist if the subscription is to be satisfied.

  For SCADA applications, the SCADA port receives the publications. You do not have to explicitly specify the port.

When the publication node receives a message, it checks the subscription table to determine whether there are any subscription requests that either specify this particular node's subscription point, or match the content or topic, or both, of the message received.

For every match found, the node delivers the published message on the subscriber queue, using the optional CorrelId, if specified. If no CorrelId is specified, a fixed value is used. Each subscriber receives just one copy of each publication regardless of how many matching subscriptions that the client has.

SCADA applications use the SCADA port to publish and subscribe, and CorrelId is not applicable.

When the node has sent the publication to any subscribers that have a matching subscription, the publication is discarded, unless it is a retained publication.

Local subscriptions

Subscribers can specify a local option on registration. If they do so, their subscription registration is not forwarded to other brokers, but is held by the local broker. Any message that is published at this broker and matches the subscription is received by this subscriber, but messages published to other brokers are not normally available, unless the subscriber has also registered a global subscription with an overlapping topic and the same subscription point.

Retained publications

If retained publications are used, the subscriber can specify the following options when it registers a subscription.

- Publish on request only

  If the Publish on Request Only option is used, the broker does not send publications to the subscriber until the subscriber sends a Request Update message to the broker. The broker then sends any current retained publication that matches the subscription.

- New publications only

  Normally the broker sends the current retained publication that matches the subscription when a subscriber registers that subscription. If the subscriber uses the New Publications Only option, the broker waits until a new publication is received before sending it to the subscriber.

- Message persistence

  Send all subscription registration messages as persistent messages. All subscriptions are maintained persistently by the broker.

Brokers maintain the persistence of publications as set by the publisher, unless changed by options specified when the subscription is registered. These options are:

– Nonpersistent
– Persistent
– Persistence as queue
– Persistence as publisher (the default)

The system administrator decides which users are allowed to have publications sent persistently.

Deregistering a subscription

One or more subscriptions for a particular subscriber can be deregistered using the Deregister Subscriber command message. This is sent to the broker control queue, SYSTEM.BROKER.CONTROL.QUEUE. The message must be sent by the subscriber that registered the subscription in the first place.

There are other ways in which a subscription can be deregistered; these are listed below.

- The subscription expires because the expiry time has passed.
- A system administrator deregisters the subscription.
- If the subscriber queue is a temporary dynamic queue, and the queue is deleted (for example, when the subscriber disconnects from the queue manager), the broker deregisters the subscription automatically.

  Automatic deregistration does not occur if:

  – The temporary dynamic queue is not local (that is, it is not on the same queue manager on which the broker is running).
  – The subscriber has named a queue that is an alias of a local temporary dynamic queue.

When a subscriber application sends a message to deregister a subscription, and receives a response message to say that this has been done successfully, some publications might subsequently reach the subscriber queue if they were being processed by the broker at the same time as the deregistration. This might result in a buildup of unprocessed messages on the subscriber queue. The application can clear these unprocessed messages from the queue by repeatedly sleeping and sending an MQGET call with the appropriate CorrelId.

Similarly, if the subscriber uses a permanent dynamic queue and, when terminating, it deregisters and closes the queue with the PurgeandDelete option, the queue might not be empty. This is because publications from the broker might not yet be committed at the time that the queue was deleted. In this case, a *Q_NOT_EMPTY* return code is issued by the MQCLOSE call. The application can avoid this problem by repeatedly sleeping and reissuing the MQCLOSE call.

Generating statistics reports

You can generate statistics reports that provide information about the performance of your brokers.

Use these statistics reports to show you where there are performance problems in your broker network. You can then change the properties of the brokers that might affect performance. These properties are described in "Performance considerations for Real-time transport" on page 30.

By default, statistics reporting is disabled. To enable statistics reporting:

1. Use the property statsInterval of the mqsichangeproperties command:

   ```
   mqsichangeproperties <broker name> -e <execution group> -o DynamicSubscriptionEngine
                        -n statsInterval -v <time interval>
   ```

   where <broker name> is the name of the broker whose statistics you want to be reported, <execution group> is the name of the execution group deployed to the broker, and <time interval> is the number of milliseconds that should separate statistics reports.

   There is a small performance overhead when statistics reporting is enabled; the smaller the time interval specified, the larger the performance overhead.
2. Restart the broker.

To disable statistics reporting, use the same procedure, but set the value of the statsInterval property to 0.

Subscribing to statistics reports

When statistics reporting is enabled for a broker, the broker publishes a statistics report at regular intervals (as determined by the value specified for the statsInterval property of the broker). The statistics reported is distributed, as a publication, to all subscribers that subscribed to the topic

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics`

where *broker name* is the name of the broker, and *execution group* is the name of the execution group that is deployed to that broker.

You can use wild cards when you subscribe to statistics reports. For example, to receive statistics reports for all brokers and all execution groups, subscribe to the topic

`$SYS/Broker/+/ExecutionGroup/+/Statistics`

Subscribers receive statistics reports only from those brokers that have been enabled to produce statistics.

The publication is a JMS Bytes Message that contains the statistics report in XML format.

Subscribing to multicast statistics reports

When statistics reporting is enabled for a broker that is enabled for multicast, the broker publishes statistics reports at regular intervals (as determined by the value specified for the statsInterval property of the broker). The statistics reports are distributed, as publications, to all subscribers that subscribed to the following topics:

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics/Multicast/Topics`

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics/Multicast/Groups`

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics/Multicast/Summary`

where *broker name* is the name of the broker, and *execution group* is the name of the execution group that is deployed to that broker.

Subscribers receive statistics reports only from those brokers that have been enabled to produce statistics.

The publications are JMS Bytes Messages that contain the statistics reports in XML format.

Part 3. Reference

Publish/subscribe

Publish/subscribe reference information is available for:
- "Special characters in topics"
- "Topic semantics and usage" on page 46
- "Sample authentication exchanges" on page 47
- "WebSphere MQ Publish/Subscribe" on page 52
- "MQRFH2 header" on page 66
- "Command messages" on page 71

Special characters in topics

A topic can contain any character in the Unicode character set. However, the following three characters have a special meaning:

The topic level separator "/".
The multilevel wild card "#".
The single-level wild card "+".

The topic level separator is used to introduce structure into the topic, and can therefore be specified within the topic for that purpose.

The multilevel wild card and single-level wild card can be used for subscriptions, but they cannot be used within a topic by the publisher of a message.

However, if a publisher uses the characters "+" or "#" together with other characters in any topic level within a topic, these characters are not treated as wild cards, and they do not have any special meaning.

The topic level separator

The topic level separator character "/" is used to provide a hierarchical structure to the topic space. It must be used by applications to separate levels within a topic tree. The use of the topic level separator is significant when the two wildcard characters are encountered in topics specified by subscribers.

Topic hierarchy is important in the administration of access control.

The multilevel wild card

The multilevel wildcard character "#" is used to match any number of levels within a topic. For example, using the example topic tree shown above, if you subscribe to "USA/Alaska/#", you receive messages on topics "USA/Alaska" and "USA/Alaska/Juneau".

The multilevel wild card can represent zero or more levels. Therefore, "USA/#" can also match the singular "USA", where # represents zero levels. The topic level separator is meaningless in this context, because there are no levels to separate.

The multilevel wild card can be specified only on its own or next to the topic level separator character. Therefore, "#" and "USA/#" are valid topics where the "#" character is treated as a wild card. However, although "USA#" is also a valid topic,

the ″#″ character is not regarded as a wild card and does not have any special meaning. See “When wild cards are not wild” for more information.

The single-level wild card

The single-level wildcard character ″+″ matches one, and only one, topic level. For example, ″USA/+″ matches ″USA/Alabama″, but not ″USA/Alabama/Auburn″. Also, because the single-level wild card matches only a single level, ″USA/+″ does not match ″USA″.

The single-level wild card can be used at any level in the topic tree, and in conjunction with the multilevel wild card. The single-level wild card must be specified next to the topic level separator, except when it is specified on its own. Therefore, ″+″ and ″USA/+″ are valid topics where the ″+″ character is treated as a wild card. However, although ″USA+″ is also a valid topic, the ″+″ character is not regarded as a wild card and does not have any special meaning. See “When wild cards are not wild” for more information.

When wild cards are not wild

The wildcard characters ″+″ and ″#″ have no special meaning when they are mixed with other characters (including themselves) in a topic level.

This means that topics that contain ″+″ or ″#″ together with other characters in a topic level can be published.

For example, consider the following two topics:

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

In the first example, the characters ″+″ and ″#″ are treated as wild cards and are therefore not valid in a topic that is to be published.

In the second example, the characters ″+″ and ″#″ are not treated as wild cards and therefore the topic can be both published and subscribed to.

Topic semantics and usage

When you build an application, the design of the topic tree should take into account the following principles of topic name syntax and semantics:

- Topic names are case sensitive.

  For example, ″ACCOUNTS″ and ″Accounts″ are two different topics.
- Topic names can include the space character.

  For example, ″Accounts payable″ is a valid topic.
- A topic level can be an empty string, but this is not recommended.

  For example, ″a//c″ is a three-level topic name with an empty middle level.
- A leading ″/″ creates a distinct topic.

  For example, ″/USA″ is different from ″USA″ and ″/USA’ matches ″+/+″ and ″/+″, but not ″+″.
- Do not include the null character (Unicode \x0000) in any topic.

- The wildcard characters ″+″ and ″#″ are not treated as wild cards if they are mixed with any other characters (including themselves but excluding the topic level separator ″/″) within a topic level.

The following principles apply to the construction and content of a topic tree:

- There is no limit to the number of levels in a topic tree.
- There is no limit to the length of the name of a level in a topic tree.
- There can be any number of ″root″ nodes; that is, there can be any number of topic trees. These are defined below the root ″″, which is the root of all root nodes. It is referred to as ″topicRoot″, although there is no corresponding topic name. Applications cannot publish or subscribe to this virtual root.
- The topic trees with roots of ″$SYS″ and ″$ISYS″ are reserved for use by WebSphere Event Broker only.

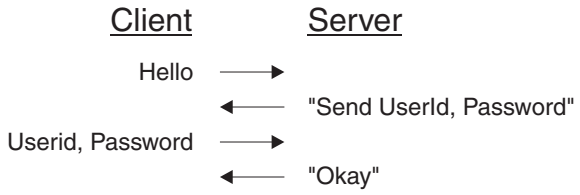
  If you are using topic-based security, only brokers can publish messages on these topics, and only brokers can subscribe to messages with a topic of ″$ISYS″, regardless of the content of the topic Access Control Lists (ACLs) that are defined.

  For more details about topic-based security and ACLs, see Topic-based security.

Sample authentication exchanges

Look at “Simple telnet-like password authentication” and “Simple mutual challenge-response password authentication” for examples of authentication exchanges.

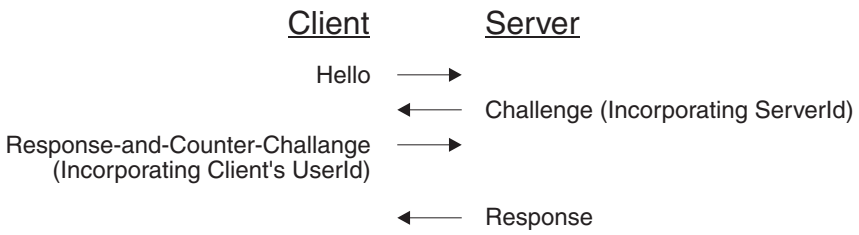
Simple telnet-like password authentication



This diagram shows the following exchange of messages between a client and a server in a simple telnet-like password authentication.

1. The client sends ″Hello″.
2. The server replies ″Send UserId, Password″.
3. The client sends a user ID and password.
4. The server concludes the exchange of messages with ″Okay″.

Simple mutual challenge-response password authentication



Statistics reports

The statistics data is published by the broker in XML format.

The report is split into three sections:

- Broker
- Client
- Neighbor

The content of each section is described below.

Broker Statistics are:

Broker
The name of the broker that generated the statistics report.

Execution Group
The name of the execution group running on the broker.

Client Count
The total number of clients that are connected to the broker.

Neighbor Count
The total number of neighbor brokers that are connected to the broker.

Subscription Count
The number of subscriptions held by the broker.

Time Stamp
The time when the statistics report was sent.

Client Statistics, which consists of data about message throughput between the broker and the clients that are connected to the broker, are:

Bytes Queued
The number of bytes of data that are currently queued by the broker for delivery to clients.

Messages Sent
The total number of messages that the broker has delivered to its clients.

Bytes Sent
The total number of bytes that the broker has delivered to its clients.

Bytes Cut Through
The total number of bytes of data that were sent immediately to clients, without being queued internally by the broker.

Messages Received
The total number of messages that have been received by the broker from its clients.

Bytes Received
The total number of bytes that have been received by the broker from its clients.

Messages Dropped
The number of messages that have been dropped due to queue overflow, where the client was not subsequently disconnected from the broker.

Bytes Dropped
The number of bytes that have been dropped due to queue overflow, where the client was not subsequently disconnected from the broker.

Disconnect Messages Dropped
The number of messages that have been dropped due to queue overflow, where the client was subsequently disconnected from the broker.

Disconnect Bytes Dropped
The number of bytes that have been dropped due to queue overflow, where the client was subsequently disconnected from the broker.

Neighbor Statistics, which consists of data about the message throughput between the broker and any other brokers with which it has been configured as a neighbor, are:

Bytes Queued
The number of bytes of data that are currently queued by the broker for delivery to neighboring brokers.

Messages Sent
The total number of messages that the broker has delivered to neighboring brokers.

Bytes Sent
The number of bytes that the broker has delivered to neighboring brokers.

Bytes Cut Through
The total number of bytes of data that were sent immediately to neighboring brokers, without being queued internally by the broker.

Messages Received
The total number of messages that have been received by the broker from neighboring brokers.

Bytes Received
The total number of bytes that have been received by the broker from neighboring brokers.

Messages Dropped
The number of messages dropped due to queue overflow, where the neighboring broker was not subsequently disconnected from the broker.

Bytes Dropped
The number of bytes dropped due to queue overflow, where the neighboring broker was not subsequently disconnected from the broker.

Disconnect Messages Dropped
The number of messages dropped due to queue overflow, where the neighboring broker was subsequently disconnected from the broker.

Disconnect Bytes Dropped
The number of bytes dropped due to queue overflow, where the neighboring broker was subsequently disconnected from the broker.

Multicast statistics reports

Multicast statistics data is published by the broker in XML format.

The multicast reports contain the following fields:

msgsSent
The total number of multicast messages that have been sent by the broker since it was started.

bytesSent
The total number of bytes in multicast messages that have been sent by the broker since it was started.

queueDepth
The total number of bytes of data that are currently queued by the broker for multicast delivery.

bytesResent
The total number of bytes that have been resent in multicast messages since the broker was started.

The content of the three multicast reports is described below.

Summary report

The subscription was to topic
`$SYS/Broker/<broker name>/ExecutionGroup/<execution group>/Statistics/Multicast/Summary`

Broker
The name of the broker that generated the statistics report.

Execution Group
The name of the execution group running on the broker.

Time Stamp
The time when the statistics report was sent.

Topic Name
The value of this field is the wildcard character "#".

Messages Sent
The total number of multicast messages that have been sent by the broker since it was started.

Bytes Sent
The total number of bytes in multicast messages that have been sent by the broker since it was started.

Bytes Queued
The total number of bytes of data that are currently queued by the broker for multicast delivery.

Bytes Resent
The total number of bytes that have been resent in multicast messages since the broker was started.

Group Address
The value of this field is the wildcard character "*".

Report by multicast group

If the subscription was to topic
`$SYS/Broker/<broker name>/ExecutionGroup/<execution group>/Statistics/Multicast/Groups`

Broker
The name of the broker that generated the statistics report.

Execution Group
The name of the execution group running on the broker.

Time Stamp
The time when the statistics report was sent.

Group Name
The name of a multicast group.

Group Address
The internet address of the multicast group.

Messages Sent
The total number of multicast messages that have been sent by the broker since it was started, for the multicast group named.

Bytes Sent
The total number of bytes in multicast messages that have been sent by the broker since it was started, for the multicast group named.

Bytes Queued
The total number of bytes of data that are currently queued by the broker for multicast delivery, for the multicast group named.

Bytes Resent
The total number of bytes that have been resent in multicast messages since the broker was started, for the multicast group named.

The last six fields listed above are repeated for each multicast group currently being used.

Report by multicast topic

If the subscription was to topic
`$SYS/Broker/<broker name>/ExecutionGroup/<execution group>/Statistics/Multicast/Topics`

Broker
The name of the broker that generated the statistics report.

Execution Group
The name of the execution group running on the broker.

Time Stamp
The time when the statistics report was sent.

Topic name
The name of the multicast topic.

Messages Sent
The total number of multicast messages that have been sent by the broker since it was started, for the multicast group named.

Bytes Sent
The total number of bytes in multicast messages that have been sent by the broker since it was started, for the multicast group named.

Bytes Queued
The total number of bytes of data that are currently queued by the broker for multicast delivery, for the multicast group named.

Bytes Resent
The total number of bytes that have been resent in multicast messages, since the broker was started, for the multicast group named.

Group Name
The name of the multicast group that contains the topic.

The last six fields listed above are repeated for each multicast topic currently being used.

WebSphere MQ Publish/Subscribe

WebSphere MQ Publish/Subscribe application support differs in some way from the publish/subscribe application support provided by WebSphere Event Broker. These differences are described in "Product differences."

Product differences

The differences in the publish/subscribe support provided by WebSphere MQ and WebSphere Event Broker are described in the following topics:

- "Message formats"
- "Streams" on page 55
- "Stream authority" on page 59
- "Topics" on page 60
- "Wildcard characters" on page 61
- "Default topic routing" on page 62
- "Retained publications" on page 63
- "Metatopics" on page 63
- "Subscription points" on page 63
- "Content-based filtering" on page 64
- "Throughput" on page 65

Message formats

Client applications that are developed for WebSphere Event Broker should use the MQRFH2 message header. These applications can then use all the function provided by WebSphere Event Broker.

Existing WebSphere MQ Publish/Subscribe applications that use the MQRFH message header are also supported by WebSphere Event Broker, but function is limited to that provided by WebSphere MQ Publish/Subscribe.

WebSphere MQ Publish/Subscribe does not support the MQRFH2 format. Clients that are connected to WebSphere MQ Publish/Subscribe brokers must use the MQRFH format.

However, client applications that need to communicate with one another using publish/subscribe can do so regardless of the message format that they use. WebSphere Event Broker provides automatic conversion to ensure that a subscriber receives messages in the correct format.

The following table shows the mapping between equivalent fields in the MQRFH and MQRFH2 message headers:

MQRFH field name	**MQRFH2 field name**
MQPSCommand	Command
MQPSDelOpts	DelOpt
MQPSPubOpts	PubOpt
MQPSPubTime	PubTime
MQPSQMgrName	QMgrName
MQPSQName	QName
MQPSRegOpts	RegOpt
MQPSSeqNum	SeqNum
MQPSTopic	Topic

All the MQRFH2 fields that are shown in the table are contained in a <psc> folder.

Field names that are not shown in the table do not have a common meaning, or are valid in only one of the two header formats. Field names that are not recognized, or not appropriate to the other format, are not copied. For example, the following name-value area of an MQRFH:

```
MQPSCommand Publish
MQPSPubOpts RetainPub
MQPSStreamName SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic "Sport/Soccer/State/LatestScore/Team1 Team2"
```

is converted to this MQRFH2 folder:

```
<psc>
<Command>Publish</Command>
<PubOpt>RetainPub</PubOpt>
<Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Using these mapping rules, WebSphere Event Broker makes sure that MQRFH2 publications can still be received by MQRFH subscribers, and that MQRFH publications can be received by MQRFH2 subscribers.

Content filters can be specified by MQRFH2 subscribers even if the topic that they are subscribing to is one that is published in MQRFH format by an WebSphere MQ Publish/Subscribe client, although there is some limit to compatibility. These limitations are described later in this topic.

The next table summarizes the valid options for clients that use the different message formats.

Message	Option Name	Option Value	Support
All requests (client to broker)	MQPSCommand	DeletePub DeregPub DeregSub Publish RegPub RegSub ReqUpdate	yes yes[1] yes yes yes[1] yes yes
	MQMD.Format	MQFMT_PCF MQFMT_RF_HEADER	no yes
	MQMD.Report	MQRO_PAN MQRO_NAN	yes yes
	MQMD.MsgType	MQMT_REQUEST MQMT_DATAGRAM	yes yes
	MQMD.MsgId		yes
	MQMD.CorrelId		yes[4]
	MQMD.ReplyToQ		yes
	MQMD.ReplyToQMgr		yes
	MQPSStreamName		prefixed on topic[3]
	MQPSTopic		yes
All requests except Delete Publication	MQPSQMgrName		yes
	MQPSQName		yes
	MQPSRegOpts	CorrelAsId	yes
Delete Publication	MQPSDelOpts	Local	yes[5]
Deregister Publisher[1]	MQPSRegOpts	DeregAll	yes
Deregister Subscriber	MQPSRegOpts	DeregAll	yes
Publish	MQMD fields	As specified by MQPS[2]	yes
	MQPSRegOpts	Anon	yes[7]
		Local	yes[5]
		DirectReq	yes[1]
	MQPSPubOpts	NoReg	yes[1]
		RetainPub	yes (set by publisher)
		IsRetainedPub	yes (set by broker)
		OtherSubsOnly	yes
	MQPSPubTime		yes
	MQPSSeqNum		yes
	MQPSStringData[1]		yes
	MQPSIntData[1]		yes

Message	Option Name	Option Value	Support
Register Publisher[1]	MQPSRegOpts	Anon	yes[7]
		Local	yes[5]
		DirectReq	yes[1]
Register Subscriber	MQPSRegOpts	Anon	yes[7]
		Local	yes[5]
		NewPubsOnly	yes
		PubOnReqOnly	yes
		InclStreamName	no[3]
		InformIfRet	yes
All responses (broker to client)	MQPSCompCode		new values added[6]
	MQPSReason		new values added[6]
	MQPSReasonText		new values added[6]
	MQPSCommand		command to which this is a response

Notes:
1. This option is supported for migration purposes.
2. MQPS is WebSphere MQ Publish/Subscribe.
3. The stream name parameter is effectively prefixed on the topic. The stream name can be deduced from the queue name if the property *implicitStreamNaming* of the Publication node is set.
4. The client identity is determined as the concatenation of the queue manager name, the queue name, and optionally the correlation id (when the correlation ID as identity option is set). The application identifier is thus "MQPSQMgrName:MQPSQName[:correlId]". The default values specified by WebSphere MQ Publish/Subscribe are used if these values are not present in a message.
5. The behavior of this option differs.
6. New values have been added.
7. Ignored by WebSphere Event Broker.

Streams

WebSphere MQ Publish/Subscribe uses streams primarily as a way of partitioning the topic name space. Sets of related topics can be grouped together into separate streams, allowing different security controls to be applied to different streams, and the publishing work load of the broker to be better balanced.

However, WebSphere Event Broker provides more flexible controls to achieve both of these behaviors. Therefore, the concept of a stream is supported only for MQRFH application compatibility.

The security controls of WebSphere Event Broker allow authorization to be applied to an individual topic level. Also, the publishing workload of the broker can be more easily controlled by creating additional instances of publication message flows that can serve either the same or different input queues.

WebSphere Event Broker still allows MQRFH client applications to specify an MQPSStreamName command parameter in their subscriptions and publications. However, the stream name is only used to modify the topic to preserve the partitioning characteristic of WebSphere MQ Publish/Subscribe.

When the stream name associated with a message is set to something other than SYSTEM.BROKER.DEFAULT.STREAM, the message is processed as if the topic, or topics, mentioned within the message had been prefixed with the string "$SYS/STREAM/<streamname>/". That is, a subscription to Topic1 that specifies a stream name of StreamX is processed as if the subscription had been made to topic "$SYS/STREAM/StreamX/Topic1".

MQRFH2 publishing and subscribing applications can still target stream-related topics, even though they themselves are not allowed to specify a stream name in the messages that they send to the WebSphere Event Broker broker. To do this, they must prefix the topics with the appropriate stream prefix.

For example, to subscribe to topic "IBM/Latest" that is published on stream STOCK.STREAM within the WebSphere MQ Publish/Subscribe network, an MQRFH2 subscriber must specify topic "$SYS/STREAM/STOCK.STREAM/IBM/Latest".

WebSphere MQ Publish/Subscribe allows a stream-related publication to be sent only to a queue that has the same name as the stream. However, WebSphere Event Broker allows publishing clients to send their publications to any input queue in a message flow.

MQRFH applications that explicitly specify a stream name parameter within a publication can send it to any publication queue that is serviced by the WebSphere Event Broker broker. The queue does not need to have the same name as the stream.

Be aware that the order in which publications are received might be different from what you might expect.

Each Publication node has an Implicit Stream Naming property whose default value is *true*. This default option results in behavior that is identical to that in WebSphere MQ Publish/Subscribe when an MQRFH publication does not contain an explicit stream name. If this property is *false*, and the publication does not contain an explicit stream name, SYSTEM.BROKER.DEFAULT.STREAM is assumed.

The next table summarizes the options that are available to both MQRFH and MQRFH2 client applications that publish messages, either to the default stream or to a specific WebSphere MQ Publish/Subscribe stream. The stream name *StreamX* is used to illustrate the options.

	MQRFH publisher		**MQRFH2 publisher**	
	default stream	StreamX	default stream	StreamX

	MQRFH publisher		MQRFH2 publisher	
MQRFH subscriber	S1,P1	S2,P2	S1,P3	S2,P4
MQRFH2 subscriber	S3,P1	S4,P2	S3,P3	S4,P4

Subscriber notes:
- **S1** Subscriber subscribes either without a stream name or with stream name ″SYSTEM.BROKER.DEFAULT.STREAM″.
- **S2** Subscriber subscribes with stream name ″StreamX″.
- **S3** Subscriber subscribes on topic without adding ″$SYS/STREAM/<streamname>/″.
- **S4** Subscriber subscribes on topic prefixed with ″$SYS/STREAM/StreamX/″.

Publisher notes:
- **P1** Publisher publishes on any queue specifying stream name ″SYSTEM.BROKER.DEFAULT.STREAM″. or publishes without specifying a stream name on any queue with the Implicit Stream Naming property set to false.
- **P2** Publisher publishes on any queue specifying stream name ″StreamX″, or publishes without specifying a stream name on queue ″StreamX″ with the Implicit Stream Naming property set to true.
- **P3** Publisher publishes on any queue without adding the prefix ″$SYS/STREAM/<Stream>/″ to the topic.
- **P4** Publisher publishes on any queue and adds the prefix ″$SYS/STREAM/StreamX/″ to the topic.

Note: The ″$SYS/STREAM/<streamname>/″ prefix is removed from all topics in an MQRFH2 publication when it is delivered to an MQRFH subscriber.

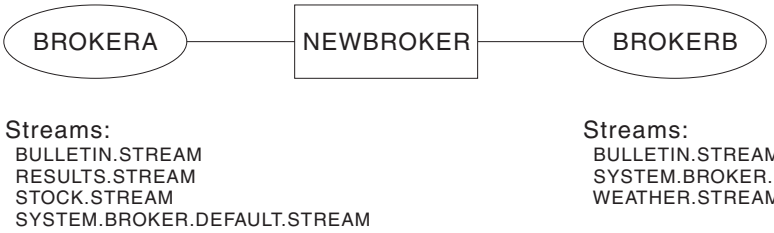
Streams and neighboring brokers:

In an WebSphere MQ Publish/Subscribe network a broker might not support the same set of streams as its neighbors. If a broker does not support a stream that is supported by one of its neighboring brokers, publications associated with that stream are not available to clients at that broker.

When a WebSphere Event Broker broker is added to the network, it supports all the streams that are supported by its neighboring WebSphere MQ Publish/Subscribe brokers. This means that clients of the WebSphere Event Broker broker can target publications for any stream supported by any of its WebSphere MQ Publish/Subscribe neighbors.

However, to make these publications available, you must define the stream queues, and define and deploy the message flows that support them, to the WebSphere Event Broker broker.

The effects of adding a WebSphere Event Broker broker into a multi-stream WebSphere MQ Publish/Subscribe environment are shown in the following example:



The WebSphere Event Broker broker, NEWBROKER, has been used to join WebSphere MQ Publish/Subscribe brokers, BROKERA, and BROKERB.

The default stream queue SYSTEM.BROKER.DEFAULT.STREAM is always supported by every broker in a WebSphere MQ Publish/Subscribe network, and must be defined for every WebSphere Event Broker broker in a heterogeneous network. At each broker, you must also define and deploy a message flow to service this queue.

When a WebSphere Event Broker broker is integrated into a WebSphere MQ Publish/Subscribe network, and links two or more WebSphere MQ Publish/Subscribe brokers that share common streams, you must define the common stream queues, and define and deploy the message flows that service them, to the WebSphere Event Broker broker.

For example, the WebSphere Event Broker broker NEWBROKER shown in the previous figure must have a stream queue defined for BULLETIN.STREAM. It must also have a message flow defined and deployed to provide a publication service for that queue.

You need to define stream queues and associated message flows to the WebSphere Event Broker broker for the other streams shown in the figure only if one of its WebSphere MQ Publish/Subscribe neighbors might send a message to one of these queues. A message is sent if one of the following events occur:

1. A subscription to a publication on one of these streams is registered by a client of the WebSphere Event Broker broker.
2. A DeletePublication command for the stream is issued by a client anywhere within the broker network.

If you are not sure whether the above cases might occur, create stream queues and message flows in the WebSphere Event Broker broker for every stream that is supported by a WebSphere MQ Publish/Subscribe neighbor. If you do not do this, you might see the following results:

- Messages sent from WebSphere MQ Publish/Subscribe brokers are put to the dead-letter queue (DLQ) of the WebSphere Event Broker broker if the stream queue does not exist on that broker.
- Messages build up on stream queues on the WebSphere Event Broker broker if the stream queue exists but no message flow is deployed to service it.

Streams and migration:

When a WebSphere MQ Publish/Subscribe broker is migrated to a WebSphere Event Broker broker (using the migmqbrk command), the streams that are supported at the time of the migration are exactly replicated in the WebSphere Event Broker broker.

No changes can be made subsequently; that is, no streams can be added or removed from this replicated set.

Migration is not complete until you have created and deployed message flows that process all these streams.

Stream authority

In WebSphere MQ Publish/Subscribe, all publish and subscribe authority checks are performed against the stream queue:

- Publishing applications require the authority to put messages to the stream queue.
- The WebSphere MQ Publish/Subscribe broker checks the authority of subscribing applications that want to browse the stream queue.
- Subscribing applications require the authority to put messages to the queue that it nominated to receive its publications.

The same check is made by WebSphere Event Broker brokers, but the subscribe authority (browse) is no longer checked.

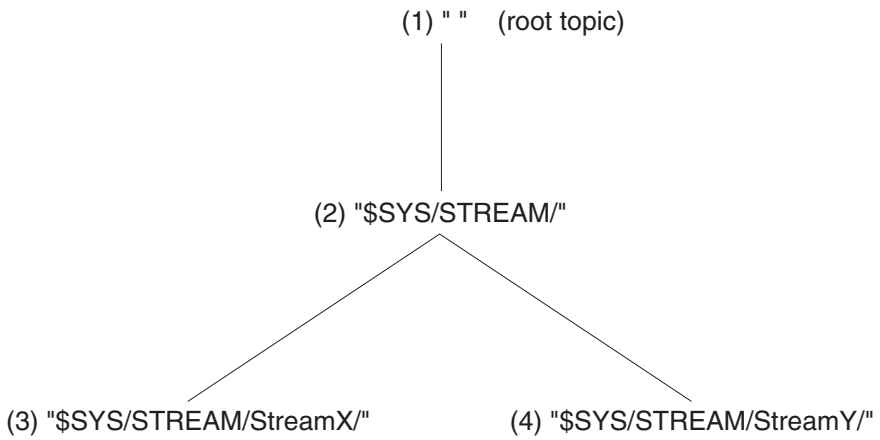
Instead, WebSphere Event Broker allows both publish and subscribe access to be defined in a hierarchical manner right down to an individual topic level. To do this, use the workbench to create access control lists (ACLs).

Before you migrate a WebSphere MQ Publish/Subscribe broker to a replacement WebSphere Event Broker broker, or before you migrate your WebSphere MQ Publish/Subscribe applications to run on WebSphere Event Broker, you must consider the security implications:

- Publishing applications are subject to the same checks even if your broker is not running with topic security enabled, because the authority to put a message to the stream or publication queue continues to be checked by WebSphere MQ.

  However, stream publications can be processed by WebSphere Event Broker on any input queue, because publishers no longer need to put to a queue with the same name as the stream. You should therefore set up equivalent ACLs for all streams using their corresponding topic level qualifiers
- The WebSphere Event Broker broker does not check that subscribing applications have the authority to browse the stream queue. Instead, WebSphere Event Broker models streams by prefixing all topics that are not part of the default stream with the unique prefix, $SYS/STREAM/<streamname>/. This maintains the partitioning characteristics of streams and allows stream-specific ACLs to be set up. Topics in the default stream are not altered by the broker. Therefore, the root topic can be used to specify the authorities for default stream topics.

The diagram below shows the stream authorities that are required. The example assumes that you have updated the default ACL on the topic root for principal PublicGroup with authority for publish, subscribe, and persistent delivery all set to *deny*.



Using this example, assume that the following groups are defined:
- PDefault: the group of users authorized to publish on the default stream
- SDefault: the group of users authorized to subscribe to the default stream
- PStreamX: the group of users authorized to publish on StreamX
- SStreamX: the group of users authorized to subscribe to StreamX
- PStreamY: the group of users authorized to publish on StreamY
- SStreamY: the group of users authorized to subscribe to StreamY

You must grant and deny authorities by setting up ACLs as follows:

1. PDefault must be granted publish authority on the root; SDefault must be granted subscribe authority on the root.
2. PDefault must be denied publish authority on $SYS/STREAM/; SDefault must be denied subscribe authority on $SYS/STREAM/.

   These settings ensure that publishers and subscribers on the default stream cannot automatically publish on, or subscribe to, other streams; an ACL must be defined that explicitly overrides that setting.
3. PStreamX must be granted publish authority on $SYS/STREAM/StreamX/, SStreamX must be granted subscribe authority on $SYS/STREAM/StreamX/.

   These settings override any setting on parent topics and limit publish and subscribe activity to users within these specific groups.
4. PStreamY must be granted publish authority on $SYS/STREAM/StreamY/, SStreamY must be granted subscribe authority on $SYS/STREAM/StreamY/.

   These settings override any setting on parent topics and limit publish and subscribe activity to users within these specific groups.

If you want to set up exceptions to this situation, you need to introducing an ACL at the appropriate point. For example, if you wanted to grant authority to publishers to the default stream (PDefault) to publish on StreamX, you must create an explicit ACL at point (3) to grant that authority, thus overriding the denial at point (2). In this scenario, users in PDefault still could not publish on StreamY.

Topics

In WebSphere MQ Publish/Subscribe, all publications must be tagged with an arbitrary character string called a topic. This defines the subject matter of the publication.

WebSphere MQ Publish/Subscribe recommends, though this is not enforced, that topic strings are structured into a number of fields or levels using the forward slash, "/", as a delimiter.

WebSphere Event Broker publications also have a topic associated with them, and the topic structure is delimited by the forward slash character.

Therefore, if your existing applications follow the WebSphere MQ Publish/Subscribe recommendation, they are better positioned to exploit the function provided by WebSphere Event Broker, which allows the structure of the topic to be externalized.

WebSphere Event Broker allows you to control users' authority to publish on, and subscribe to, any topic at any level within the topic structure.

Wildcard characters

Wildcard characters can be used by subscribing applications to broaden the scope of publications that they register an interest in. By specifying a wildcard character, the subscriber is specifying a general pattern of the topics they are interested in, rather than an explicit topic.

Wildcard characters are used by both WebSphere MQ Publish/Subscribe and WebSphere Event Broker. However, WebSphere Event Broker provides a different set of wildcard characters that allow a more extensive and flexible use of wildcard characters by subscribers.

- WebSphere MQ Publish/Subscribe wildcard characters are:
  - An asterisk (*); this matches zero or more characters.
  - A question mark (?); this matches exactly one character.
  - The percent sign (%); this can be used as an escape character to use an "*", a "?", or a "%" character within a topic.
- WebSphere Event Broker wildcard characters are:
  - The multilevel wild card (the character #); this matches any number of levels at the start or end of the topic.
  - The single-level wild card (the character +); this matches a single level within the topic.

  The characters used are:

The full range of function of the WebSphere Event Broker wildcard characters is only available to MQRFH2 clients. Subscriptions that are made by MQRFH clients to WebSphere Event Broker brokers for topics that contain either of the WebSphere Event Broker wildcard characters are rejected with the reason code MQRCCF_TOPIC_ERROR.

Applications that use MQRFH and connect to WebSphere MQ Publish/Subscribe brokers in a heterogeneous network should therefore not publish on, or subscribe to, topics that contain either the multilevel wild card (#) or single-level wild card (+) characters. WebSphere MQ Publish/Subscribe brokers do not police this; if your applications specify any WebSphere Event Broker wildcard characters in topics when they publish or register a subscription in a heterogeneous broker network, these publications and subscriptions are ignored by WebSphere Event Broker brokers within the network. You are therefore strongly advised to review and if necessary change the topics being used within a WebSphere MQ Publish/Subscribe implementation before adding a WebSphere Event Broker broker to the network.

When applications that use MQRFH2 use the WebSphere Event Broker wildcard characters to target multiple publications from within the WebSphere MQ Publish/Subscribe network, wildcard mapping is performed. In most cases, the broker replaces both the multilevel wild card and the single-level wild card characters with an asterisk. This does not provide an exact match for either of the WebSphere Event Broker wildcard characters, but ensures that a superset of the required publications are sent to the WebSphere Event Broker broker. The WebSphere Event Broker broker evaluates the ″#″ and ″+″ wildcard characters to match the correct publications.

For example, the topic ″employee/+/development″ is propagated as ″employee/*/development″ to a WebSphere MQ Publish/Subscribe neighbor. This might cause redundant publications to be sent to the WebSphere Event Broker broker from its WebSphere MQ Publish/Subscribe neighbor. However, none of these would be sent to the original client when the WebSphere Event Broker evaluates the original subscription.

The exception to this is a subscription to the topic ″+″ which is never propagated; it cannot be represented as an ″*″ because this is the topic that is propagated if a subscription to topic ″#″ is made at the WebSphere Event Broker broker.

Do not specify WebSphere MQ Publish/Subscribe wildcard characters in MQRFH2 client subscriptions. If you do specify one or more, they are assumed by WebSphere Event Broker to be part of the topic, and are therefore prefixed by the escape character (%) before the subscription is sent on to a WebSphere MQ Publish/Subscribe neighbor.

For example, if your MQRFH2 client subscribes with a topic of ″USA/Alaska*/Juneau?″, this is modified and passed to a WebSphere MQ Publish/Subscribe broker neighbor as ″USA/Alaska%*/Juneau%?″.

If an application using MQRFH connects to a WebSphere Event Broker broker, WebSphere Event Broker emulates the behavior of the WebSphere MQ Publish/Subscribe wildcard characters * and ? using a mixture of its own wildcard characters and filter expressions. Existing MQRFH applications that subscribe to a WebSphere Event Broker broker therefore receive the same publications as they would receive if they subscribe to a WebSphere MQ Publish/Subscribe broker.

Default topic routing

In WebSphere Event Broker, the Topic property of the MQInput node can be used to route messages that do not contain publish/subscribe parameters. This feature does not apply to MQRFH subscribers.

MQRFH subscribers expect to receive publications, with a well-formed MQRFH header, from both WebSphere MQ Publish/Subscribe and WebSphere Event Broker clients.

In the latter case, the original MQRFH2 header is converted as described earlier in this topic. However, if the message does not contain publish/subscribe information in either an MQRFH or an MQRFH2 header, the default topic is not used to send publications to an MQRFH subscriber.

Retained publications

In WebSphere MQ Publish/Subscribe, retained publications are stored on the queue manager with the same persistence as the input message. This means that when the queue manager is restarted, nonpersistent retained publications are deleted and persistent publications are retained.

In WebSphere Event Broker, retained publications are held in a database and are always preserved across restarts of any of the resources, regardless of whether the input publication is persistent.

Metatopics

WebSphere MQ Publish/Subscribe brokers provide information about publishers and subscribers via a special set of topics called metatopics. Metatopics start with the ″MQ/S/″ or ″MQ/SA/″ prefix, and are subscribed to by two categories of applications, administration programs and clients.

WebSphere Event Broker does not provide equivalent metatopics, and therefore a program (administration or client) that subscribes to WebSphere MQ Publish/Subscribe metatopics cannot work with a WebSphere Event Broker broker. However, WebSphere Event Broker does publish information about subscription events using its own set of system topics.

The following considerations apply to the two categories of application in the WebSphere Event Broker environment.

Administration programs (for example, the **amqspsd** sample) use WebSphere MQ Publish/Subscribe metatopics to display subscription information. This information is provided by WebSphere Event Broker in the workbench, which allows subscriptions to be viewed and deleted throughout the broker network.

Applications use messages published on WebSphere MQ Publish/Subscribe metatopics to, for example, request information about their own current subscriptions.

A client program can subscribe to WebSphere Event Broker system topics and process the event publications.

WebSphere Event Broker does not provide a topic that reports all the current subscriptions for a particular topic or client, but does publish whenever subscriptions are added or removed. This information is published as event information whereas WebSphere MQ Publish/Subscribe metatopics are published as state information.

Subscription points

Subscriptions points are a feature provided by WebSphere Event Broker that can be used to make information associated with a particular topic available in a number of different formats.

For example, stock prices might be published with a default currency of dollars, but might be required by subscribers expressed in other currencies.

This can be achieved by defining additional paths through the message flow that take each publication and convert the dollar stock price into another currency (for example, euros), before it is passed to its Publication node.

Each additional currency must be associated with a different subscription point and therefore a Publication node. The original publication in dollars is associated with the default subscription point.

Subscribers can then subscribe to stock prices using a combination of topic and the subscription point that provides the data in the correct currency.

Subscription points are not supported by WebSphere MQ Publish/Subscribe. You must therefore be careful if you use them in a heterogeneous network. In particular, be aware that publications can only pass between WebSphere Event Broker and WebSphere MQ Publish/Subscribe brokers on the default subscription point.

All topics that are published in a WebSphere MQ Publish/Subscribe broker domain are on the default subscription point. These topics are only available to MQRFH2 subscribers that subscribe to the topics without specifying a subscription point (that is, are using the default subscription point).

Similarly, clients at WebSphere MQ Publish/Subscribe brokers can only subscribe to topics that are published on the default subscription point at WebSphere Event Broker brokers (at Publication nodes that do not have a subscription point set).

Content-based filtering

WebSphere Event Broker supports content-based filtering of publications. This allows an MQRFH2 subscriber to restrict the messages that it wants to receive.

When an MQRFH2 client registers a subscription with the local broker, it can specify a filter to be applied to the content of fields within each publication message.

When an MQRFH2 subscriber subscribes to MQRFH publications within the WebSphere MQ Publish/Subscribe part of a mixed broker network, all MQRFH publications are converted to MQRFH2 format by the broker before delivery to the MQRFH2 client

An MQRFH2 subscriber can also request that some content-based filtering is performed on the MQRFH publications that they are subscribing to. This can be done only if the body of the publication is in a format that can be parsed by the broker; that is, if it can be interpreted by one of the broker's default parsers. For example, messages in XML or MQPCF format can be processed in this way.

If you want to make full use of content-based filtering, you must convert publications into MQRFH2 format. Then all messages that are defined in the message repository can be interpreted by the brokers' parsers.

MQRFH clients cannot specify a content filter.

Throughput

In WebSphere MQ Publish/Subscribe, a single thread processes publications on each of the stream queues. This guarantees the order in which publications are processed from the queue.

When you consider throughput for publications in a WebSphere Event Broker broker domain, you must also consider the importance of the order in which messages are published. Techniques that increase throughput do not necessarily guarantee order.

WebSphere Event Broker supports two options that increase throughput:

1. You can configure the message flow with additional threads by setting the Additional Instances property of the MQInput node. This property causes the broker to schedule additional threads to read messages from the input queue, thus allowing publications from that queue to be processed concurrently by the broker. You must ensure that the stream (input) queue has the share attribute set (WebSphere MQ Publish/Subscribe required stream queues to have noshare set).

   If multiple threads process messages from a single queue, publications are not guaranteed to be delivered to subscribers in the order in which they are placed on the input queue. However, WebSphere Event Broker provides a method of allowing publications to be processed concurrently, while still maintaining the required sequence.

   Set the Order Mode property of the MQInput node to the value *By User ID*. This ensures the order of delivery of publications sent to the broker by a given user. When this property is set, the processing of messages that carry a given UserIdentifier field in the MQMD is held up if any other thread servicing that message flow is currently processing a message that carries the same UserIdentifier.

   The benefits of running additional instances of the message flow are negated if all publishing applications are running under the same user ID. This might be the case for publishing applications connected to a queue manager remote to the broker's queue manager. Messages from these remote publishers arrive at the broker on a channel that might have been set up to insert the channel program's user ID instead of the originating client's user ID. Refer to the *WebSphere MQ Intercommunications* book for more information about how to set the PUTAUT channel attribute to change the default channel behavior.
2. You can configure one or more additional message flows (not instances) that read publications from different queues. You must also update some of your publishing applications to publish to the new queue (or queues). This has the effect of splitting the stream, and therefore spreading the workload.

   If you decide to increase throughput using this method, you must consider the impact this has on the order in which publications are delivered. In particular, you must ensure that the publisher applications are split with respect to the topics they are publishing to ensure that order can be maintained for each topic, if this is important. If your applications publish to different queues (message flows) on the same topic, order cannot be guaranteed.

   If you update the publisher applications to send publications to a new queue that has a name different from the stream on which they are publishing, you must also update these applications to explicitly include the stream name within their publications using the MQPSStreamName parameter.

   Publishing applications that specify a stream parameter do not need to be modified, as this parameter takes precedence. However, if publishing

applications do not specify the stream parameter, the behavior is determined by the setting of the Implicit Stream Naming property of the publication node in the message flow:

- If the property is set to *false*, the default stream is assumed.
- If the property is set to *true*, the stream name is assumed to be the same as the name of the stream input queue.

MQRFH2 header

The MQRFH2 header is used to pass messages to and from a message broker belonging to WebSphere Event Broker. In a message, the MQRFH2 header follows the WebSphere MQ message descriptor (MQMD) and precedes the message body, if present.

Other headers, such as the IMS/ESA or CICS bridge headers, are allowed either before or after the MQRFH2 header, but before the message body.

If you are using the Message Queuing Interface (MQI) to write application programs you need to understand the structure and content of the MQRFH2 header.

For more information, refer to:
- "MQRFH2 structure"
- "Message service folders" on page 70

Multiple MQRFH2 headers

A message can have more than one MQRFH2 header.

For example, if an application forwards a message, including its header, to another application, a second MQRFH2 header precedes the header in the message being forwarded.

- Attributes that describe the body of the message, such as the domain, set, type, and format, or the character set ID and encoding, are taken from the *last* MQRFH2 header, which is immediately in front of the body of the message.
- Anything else, such as the topic for a publish/subscribe message, is taken from the *first* MQRFH2 header.

MQRFH2 structure

The MQRFH2 header contains information about the structure of a message, and its intended consumers, to enable a message broker to process the message and deliver or publish the message to those consumers.

The value 'MQRFH2 ' should be put in the **Format** field of the preceding header (usually the MQMD). The constant *MQFMT_RF_HEADER_2* is defined with this value.

For the C programming language, the constant *MQFMT_RF_HEADER_2_ARRAY* is also defined. This constant has the same value as *MQFMT_RF_HEADER_2*, but it is an array of characters, not a character string.

The character set and encoding of the fields in the MQRFH2 header are as follows:

- Fields other than **NameValueData** are in the character set and encoding defined by the fields **CodedCharSetId** and **Encoding** in the header structure that precedes the MQRFH2 header, or by the same fields in the MQMD structure if the MQRFH2 header is at the start of the application message data. The character set should be one that has single-byte characters for the characters that are valid in queue names.
- **NameValueData** is in the character set defined by the **NameValueCCSID** field. Note that not all Unicode character sets are valid for **NameValueCCSID**; see the description of **NameValueCCSID** for details.

  Some character sets have a representation that is dependent on the encoding. If **NameValueCCSID** defines one of these character sets, **NameValueData** must be in the same encoding as the other fields in the MQRFH2 header.
- The user data (if any) that follows **NameValueData** can be in any supported character set (single-byte, double-byte, or multi-byte), and in any supported encoding.

The MQRFH2 header contains the following fields:

Field Name	Description	Details
StrucId	Structure identifier	The value must be `MQRFH_STRUC_ID`, which is the identifier for the rules and formatting header structure,. For the C programming language, the constant `MQRFH_STRUC_ID_ARRAY` is also defined; this constant has the same value as `MQRFH_STRUC_ID`, but it is an array of characters, not a character string.
Version	Structure version number	The value must be `MQRFH_VERSION_2`, which is the Version-2 rules and formatting header structure.
Struclength	Total length of MQRFH2 (including **NameValueData**)	The initial value of this field is `MQRFH_STRUC_LENGTH_FIXED_2`, which is the length of the fixed part of the MQRFH2 header structure. This is the length in bytes of the MQRFH2 header structure, including any **NameValueLength** and **NameValueData** fields at the end of the structure. There might be more than one pair of these fields at the end of the structure, in the sequence: length1, data1, length2, data2, ... The length of any user data that follows the last **NameValueData** field at the end of the structure is not included in **StrucLength**. **Note:** If **Struclength** is not a multiple of four, problems might occur with the data conversion of user data in some operating system environments.

Field Name	Description	Details
Encoding	Numeric encoding of data that follows **NameValueData**	The initial value of this field is MQENC_NATIVE. This field specifies how numeric values in any data that follows the last **NameValueData** field are represented. This applies to binary integer data, packed decimal integer data and floating-point data.
CodedCharSetId	Character set identifier of data that follows **NameValueData**	The initial value of this field is MQCCSI_INHERIT, which means that the character set identifier is the same as that of the current structure. This field identifies the coded character set for any character strings in the data that follows the last **NameValueData** field.
Format	Format name of data that follows **NameValueData**	The initial value of this field is **MQFMT_NONE**. This field specifies the format name of any data that follows the last **NameValueData** field. The name should be padded with blanks to the length of the field. **Note:** Do not use a null character to terminate the name before the end of the field; the queue manager does not change to a blank character the null character, or any characters that follow the null character, in the MQRFH2 header. **Note:** Do not specify a name with leading or embedded blank characters.
Flags	Flags	The initial value of this field is MQRFH_NONE, which means that there are no flags.

Field Name	Description	Details
NameValueCCSID	Character set identifier of **NameValueData**	The initial value of this field is 1208, which means that the UTF-8 coded character set is used. This field identifies the coded character set for data in the **NameValueData** field. This is different from the character set for other character strings in the MQRFH2 header structure, and might be different from the character set for any character data that follows the last **NameValueData** field. **NameValueCCSID** must have one of the following values: **1200**: UCS-2 open-ended **1208**: UTF-8 **13488**: UCS-2 2.0 subset **17584**: UCS-2 2.1 subset (includes the euro symbol €) For the UCS-2 character sets, the encoding (byte order) of the **NameValueData** field must be the same as the encoding of the other fields in the MQRFH2 header structure. **Note:** Surrogate characters (X'D800' thru X'DFFF') are not supported.
The following two fields are optional, but if present they must occur as a pair. They can be repeated as a pair as many times as required. If these fields occur more than once, they must occur in the sequence ..... length1, data1, length2, data2, ...		
NameValueLength	Length of **NameValueData**	This field specifies the length, in bytes, of the **NameValueData** field that follows this field. **Note:** If **NameValueLength** is not a multiple of four, there might be a problem with the conversion of the data that follows the **NameValueData** field.
NameValueData	This is a variable-length character string containing data that is encoded using an XML-like structure	The length, in bytes, of this string is given by the **NameValueLength** field that precedes this **NameValueData** field. To avoid the problem described in the note accompanying the description of the **NameValueLength** field, either extend this field with blanks so that its length is a multiple of four, or terminate the field with a null character.

C programming language definition

The following structure is defined in the cmqc.h header file that is supplied with WebSphere MQ. The constants that are used within the **NameValueData** field are defined in the BipRfc.h header file that is supplied with WebSphere Event Broker.

```
typedef struct tagMQRFH2 {
  MQCHAR4  StrucId;         /*  Structure identifier                      */
  MQLONG   Version;         /*  Structure version number                  */
  MQLONG   StrucLength;     /*  Total length of MQRFH2 including
                                NameValueData                             */
  MQLONG   Encoding;        /*  Numeric encoding of data that follows
                                NameValueData                             */
  MQLONG   CodedCharSetId;  /*  Character set identifier of data that
                                follows NameValueData                     */
  MQCHAR8  Format;          /*  Format name of data that follows
                                NameValueData                             */
  MQLONG   Flags;           /*  Flags                                     */
  MQLONG   NameValueCCSID;  /*  Character set identifier of NameValueData   */
 } MQRFH2;
```

Message service folders

The following folder names are defined for use by WebSphere MQ products:

<mcd>
: Message content descriptor

<psc> Publish/subscribe command

<pscr>
: Publish/subscribe command response

<usr> Application (user) defined properties

Each folder is contained in a separate **NameValueData** field, each of which is preceded by a **NameValueLength** field.

Independent software vendors can choose other names for their folders. However, to avoid naming problems, we recommend that vendors prefix their chosen folder name with their internet domain name; for example, a vendor with domain name `ourcompany.com` might name its folders:

```
com.ourcompany.xxx  or com.ourcompany.ourData
```

The mcd folder

The `<mcd>` folder can contain the following elements that describe the structure of the message data in a WebSphere MQ message. They are all character strings, and are case-sensitive.

<Msd> Message service domain

Valid values are:

mrm WebSphere Event Broker MRM-managed messages.

xml The message is treated as a self-defining XML message.

xmlns The message is treated as a self-defining XML message. If your messages use XML namespaces, use *xmlns* in preference to *xml*.

xmlnsc
: The message is treated as a self-defining XML message. If your messages use XML with namespaces, use *xmlnsc* in preference to *xmlns* or *xml*, to take advantage of the compact trees that this parser produces.

mime The message uses the MIME standard for multipart messages.

idoc The message is treated as an SAP IDoc message.

none The message is treated as an opaque blob, and delivered to the recipient as is.

<Set> Message set

<Type> The name of the message type, within the specified message set, to which this message corresponds.

Note: In Version 2.1 this specifies an identifier, rather than a name, for the message type.

<Fmt> The name of the MRM physical format, within the specified message set, to which this message corresponds.

Note: In Version 2.1 this specifies an identifier, rather than a name, for the MRM physical format.

Note: The <Set>, <Type>, and <Fmt> elements are only used when <Msd> is either *mrm* or *idoc*.

The psc folder

The <psc> folder is used to convey publish/subscribe command messages to the broker.

Only one psc folder is allowed in the **NameValueData** field.

See "Command messages" for full details.

The pscr folder

The <pscr> folder is used to contain information from the broker, in response to publish/subscribe command messages.

There is only one pscr folder in a response message.

See "Broker Response message" on page 88 for full details.

The broker ignores this folder in messages that it receives from publish/subscribe applications.

The usr folder

The content model of the <usr> folder is as follows:

- Any valid XML name can be used as an element name, providing that it doesn't contain a colon
- Only simple elements, not groups, are allowed
- All elements take the default type of string, unless modified by a dt="xxx" attribute
- All elements are optional, but should occur no more than once in a folder
- An MQRFH2 instance can contain no more than one <usr> folder

Command messages

The following command messages can be sent to WebSphere Event Broker in a publish/subscribe application:

- "Delete Publication message" on page 72
- "Deregister Subscriber message" on page 73
- "Publish message" on page 78

- "Register Subscriber message" on page 80
- "Request Update message" on page 86

If you are using the Message Queue Interface (MQI) to write applications that use the publish/subscribe model, you need to understand these messages, the Broker Response message, and the message descriptor (MQMD). Refer to:

- "Broker Response message" on page 88
- "MQMD settings in command messages to the broker" on page 92
- "MQMD settings for publications forwarded by a broker" on page 93
- "MQMD settings in broker response messages" on page 94

The commands are contained in a `<psc>` folder in the **NameValueData** field of the MQRFH2 header.

The message that can be sent by a broker in response to a command message is contained in a `<pscr>` folder.

Refer to "Message service folders" on page 70 for details about the message service folders.

The descriptions of each command list the properties that can be contained in a folder. Unless otherwise specified, the properties are optional and can occur no more than once.

Names of properties are shown as `<Command>`.

Values must be in string format, for example: `Publish`.

A string constant representing the value of a property is shown in parentheses, for example: (MQPSC_PUBLISH).

String constants are defined in the header file BipRfc.h which is supplied with WebSphere Event Broker.

Delete Publication message

The Delete Publication command message is sent to a broker from a publisher, or from another broker, to tell the broker to delete any retained publications for the specified topics.

This message is sent to the input queue of a message flow that contains a *Publication* node. You must have the authority to put a message onto this queue, and to publish on the topic, or topics, that are specified in the message.

The input queue should be the queue that the original publication was sent to.

If you have the authority for some, but not all, of the topics that are specified in the *Delete Publication* command message, only those topics are deleted. A *Broker Response* message indicates which topics are not deleted.

Similarly, if a Publish command contains more than one topic, a Delete Publication command matching some, but not all, of those topics deletes only the publications for the topics that are specified in the Delete Publication command.

See “MQMD settings in command messages to the broker” on page 92 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the broker.

Properties

<Command> (*MQPSC_COMMAND***)**
The value is DeletePub(*MQPSC_DELETE_PUBLICATION*).

This property must be specified.

<Topic> (*MQPSC_TOPIC***)**
The value is a string that contains a topic for which retained publications are to be deleted. Wildcard characters can be included in the string to delete publications on more than one topic.

This property must be specified; it can be repeated for as many topics as needed.

<DelOpt> (*MQPSC_DELETE_OPTION***)**
The delete options property can take one of the following values:
Local (*MQPSC_LOCAL***)**

All retained publications for the specified topics are deleted at the local broker (that is, the broker to which this message is sent), whether they were published with the Local option or not.

Publications at other brokers are not affected.

None (*MQPSC_NONE***)**

All options take their default values. This has the same effect as omitting the DelOpt property. If other options are specified at the same time, None is ignored.

The default if this property is omitted is that all retained publications for the specified topics are deleted at all brokers in the network, regardless of whether they were published with the Local option.

Example

Here is an example of **NameValueData** for a Delete Publication command message. This is used by the sample application to delete, at the local broker, the retained publication that contains the latest score in the match between Team1 and Team2.

```
<psc>
 <Command>DeletePub</Command>
 <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
 <DelOpt>Local</DelOpt>
</psc>
```

Deregister Subscriber message

The Deregister Subscriber command message is sent to a broker from a subscriber, or to another application on a subscriber’s behalf, to indicate that it no longer wants to receive messages matching the given parameters.

This message is sent to SYSTEM.BROKER.CONTROL.QUEUE, the broker’s control queue. The user must have the necessary authority to put a message onto this queue.

See “MQMD settings in command messages to the broker” on page 92 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the broker.

An individual subscription can be deregistered by specifying the corresponding topic, subscription point and filter values of the original subscription. If any of the values were not specified (that is, they took the default values) in the original subscription, they should be omitted when the subscription is deregistered.

All subscriptions for a subscriber, or a group of subscribers, can be deregistered by using the DeregAll option. For example, if DeregAll is specified, together with a subscription point (but no topic or filter), then all subscriptions for the subscriber on the specified subscription point are deregistered, regardless of the topic and filter. Any combination of topic, filter and subscription point is allowed; if all three are specified only one subscription can match, and the DeregAll option is ignored.

The message must be sent by the subscriber that registered the subscription; this is confirmed by checking the subscriber’s user ID.

Subscriptions can also be deregistered by a system administrator. However, the subscriptions registered with a temporary dynamic queue are associated with the queue, not just the queue name. If the queue is deleted, either explicitly, or by the application disconnecting from the queue manager, it is no longer possible to use the Deregister Subscriber command to deregister the subscriptions for that queue. The subscriptions can be deregistered using the workbench, and they are removed automatically by the broker the next time that it matches a publication to the subscription, or the next time the broker restarts. Under normal circumstances, applications should deregister their subscriptions before deleting the queue, or disconnecting from the queue manager.

If a subscriber sends a message to deregister a subscription, and receives a response message to say that this was processed successfully, some publications might still reach the subscriber queue if they were being processed by the broker at the same time as the subscription was being deregistered. If the messages are not removed from the queue, there might be a buildup of unprocessed messages on the subscriber queue. If the application executes a loop that includes an MQGET call with the appropriate **CorrelId** after sleeping for a while, these messages are cleared off the queue. If you are using the SCADA Device Protocol there is an option of clean start and finish. This means that the messages are cleared away for the client.

Similarly, if the subscriber uses a permanent dynamic queue, and deregisters and closes the queue with the MQCO_DELETE_PURGE option on an MQCLOSE call, the queue might not be empty. If any publications from the broker are not yet be committed when the queue is deleted, an MQRC_Q_NOT_EMPTY return code is issued by the MQCLOSE call. The application can avoid this problem by sleeping and reissuing the MQCLOSE call from time to time.

Properties

<Command> (*MQPSC_COMMAND***)**
The value is DeregSub (*MQPSC_DEREGISTER_SUBSCRIBER*).

This property must be specified.

<Topic> (*MQPSC_TOPIC***)**
The value is a string that contains the topic to be deregistered.

This property can, optionally, be repeated if multiple topics are to be deregistered. It can be omitted if DeregAll is specified in <RegOpt>.

The topics that are specified can be a subset of those that are registered if the subscriber wants to retain subscriptions for other topics. Wildcard characters are allowed, but a topic string that contains wildcard characters must exactly match the corresponding string that was specified in the Register Subscriber command message.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT***)**
The value is a string that specifies the subscription point from which the subscription is to be detached.

This property must not be repeated. It can be omitted if a <Topic> is specified, or if DeregAll is specified in <RegOpt>. If you omit this property, the following happens:

- If you do **not** specify DeregAll, subscriptions matching the <Topic> property (and the <Filter> property, if present) are deregistered from the default subscription point.
- If you specify DeregAll, all subscriptions (matching the <Topic> and <Filter> properties if present) are deregistered from all subscription points.

Note that you cannot specify the default subscription point explicitly. Therefore, there is no way of deregistering all subscriptions from this subscription point only; you must specify the topics.

<SubIdentity> (*MQPSC_SUBSCRIPTION_IDENTITY***)**
This is a variable-length string with a maximum length of 64 characters. It is used to represent an application with an interest in a subscription. The broker maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to hold only a single identity, or an unlimited number of identities.

If the **SubIdentity** is in the identity set for the subscription then it is removed from the set. If the identity set becomes empty as a result of this, the subscription is removed from the broker, unless **LeaveOnly** is specified as a value of the **RegOpt** property. If the identity set still contains other identities then the subscription is not removed from the broker, and publication flow is not interrupted.

If **SubIdentity** is specified, but the **SubIdentity** is not in the identity set for the subscription, then the Deregister Subscriber command fails with the return code *MQRCCF_SUB_IDENTITY_ERROR*.

<Filter> (*MQPSC_FILTER***)**
The value is a string specifying the filter to be deregistered. It must match exactly, including case and any spaces, a subscription filter that has been previously registered.

This property can, optionally, be repeated if more than one filter is to be deregistered. It can be omitted if a <Topic> is specified, or if DeregAll is specified in <RegOpt>.

The filters specified can be a subset of those registered if the subscriber wants to retain subscriptions for other filters.

<RegOpt> (*MQPSC_REGISTRATION_OPTION***)**
The registration options property can take the following values:

DeregAll
(*MQPSC_DEREGISTER_ALL*)

All matching subscriptions registered for this subscriber are to be deregistered.

If you specify DeregAll:
- <Topic>, <SubPoint>, and <Filter> can be omitted.
- <Topic> and <Filter> can be repeated, if required.
- <SubPoint> must not be repeated.

If you do **not** specify DeregAll:
- <Topic> must be specified, and can be repeated if required.
- <SubPoint> and <Filter> can be omitted.
- <SubPoint> must not be repeated.
- <Filter> can be repeated, if required.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD), which must not be zero, is used to identify the subscriber. It must match the CorrelId used in the original subscription.

FullResp
()

When**FullResp** is specified all attributes of the subscription are returned in the response message, if the command does not fail.

When **FullResp** is specified **DeregAll** is not permitted in the Deregister Subscriber command. It is also not possible to specify multiple topics. The command fails with return code *MQRCCF_REG_OPTIONS_ERROR*, in both cases.

LeaveOnly
(*MQPSC_LEAVE_ONLY*)

When you specify this with a **SubIdentity** which is in the identity set for the subscription the **SubIdentity** is removed from the identity set for the subscription. The subscription is not removed from the broker, even if the resulting identity set is empty. If the **SubIdentity** value is not in the identity set the command fails with return code MQRCCF_SUB_IDENTITY_ERROR.

If **LeaveOnly** is specified with no **SubIdentity**, the command fails with return code *MQRCCF_REG_OPTIONS_ERROR*.

If neither **LeaveOnly** nor a **SubIdentity** are specified, then the subscription is removed regardless of the contents of the identity set for the subscription.

None
(MQPSC_NONE)

All options take their default values. This has the same effect as omitting the registration options property. If other options are specified at the same time, None is ignored.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the broker that associates the userid of the original registration message with the subscriber's identity and from then on

prevents any other user using that identity. If a new subscriber tries to use the same identity, the return code *MQRCCF_DUPLICATE_SUBSCRIPTION* is returned.

Any user can modify or deregister the subscription when they have suitable authority, avoiding the existing check that the userid must match that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription to be deregistered has **VariableUserId** set this must be set at deregister time to indicate which subscription is being deregistered. Otherwise, the userid of the Deregister Subscriber command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

<QMgrName> (*MQPSC_Q_MGR_NAME***)**
The value is the queue manager name for the subscriber queue. It must match the QMgrName used in the original subscription.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the broker's queue manager.

<QName> (*MQPSC_Q_NAME***)**
The value is the name of the subscriber queue. It must match the QName used in the original subscription.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank.

<SubName> (*MQPSC_SUBSCRIPTION_NAME***)**
If you specify **SubName** on a Deregister Subscriber command the **SubName** value takes precedence over all other identifier fields except the userid, unless **VariableUserId** is set on the subscription itself. If **VariableUserId** is not set, the Deregister Subscriber command succeeds only if the userid of the command message matches that of the subscription, if not the command fails with return code *MQRCCF_DUPLICATE_IDENTITY*.

If a subscription exists that matches the traditional identity of this command but has no **SubName** the Deregister Subscriber command fails with return code *MQRCCF_SUB_NAME_ERROR*. If an attempt is made to deregister a subscription that has a **SubName** using a command message that matches the traditional identity but with no **SubName** specified the command succeeds.

<SubUserData> (*MQPSC_SUBSCRIPTION_USER_DATA***)**
This is a variable-length text string. The value is stored by the broker with the subscription but has no influence on the delivery of the publication to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is there for the use of the application.

The SubUserData is returned in the Metatopic information (MQCACF_REG_SUB_USER_DATA) for a subscription if present.

Example

Here is an example of **NameValueData** for a Deregister Subscriber command message. In this example, the sample application is deregistering its subscription to

the topics which contain the latest score for all matches. The subscriber's identity, including the **CorrelId**, is taken from the defaults in the MQMD.

```
<psc>
 <Command>DeregSub</Command>
 <RegOpt>CorrelAsId</RegOpt>
 <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Publish message

The Publish command message is sent from a publisher to a broker, or from a broker to a subscriber, to publish information on a specified topic or topics.

This message is sent to the input queue of a message flow that contains a Publication node. Authority to put a message onto this queue, and to publish on the specified topic or topics, is necessary.

If the user has authority on some, but not all, topics, only those topics are published; a warning response indicates which topics are not published.

If a subscriber has any matching subscriptions, the broker forwards the Publish message to the subscriber queues defined in the corresponding Register Subscriber command messages.

See "Broker Response message" on page 88 for details of the message descriptor (MQMD) parameters needed when sending a command message to the broker, and used when a broker forwards a publication to a subscriber.

The broker forwards the Publish message to other brokers in the network that have matching subscriptions, unless it is a local publication.

Publication data, if any, is included in the body of the message. The data can be described in an `<mcd>` folder in the **NameValueData** field of the MQRFH2 header.

Properties

<Command> (*MQPSC_COMMAND***)**
The value is Publish(*MQPSC_PUBLISH*).

This property must be specified.

<Topic> (*MQPSC_TOPIC***)**
The value is a string that contains a topic that categorizes this publication. No wildcard characters are allowed.

This property must be specified, and can optionally be repeated for as many topics as needed.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT***)**
The subscription point on which the publication is published.

This property should not be included in a publication message sent to the broker but is added automatically to publication messages by the broker before those messages are sent to any appropriate subscribers. The value of the <SubPoint> property is the value of the Subscription Point attribute of the Publication node that is handling the publishing.

<PubOpt> (*MQPSC_PUBLICATION_OPTION***)**
The publication options property can take the following values:

RetainPub
(*MQPSC_RETAIN_PUB*)

The broker is to retain a copy of the publication. If this option is not set, the publication is deleted as soon as the broker has sent the publication to all its current subscribers.

IsRetainedPub
(*MQPSC_IS_RETAINED_PUB*)

(Can only be set by a broker.) This publication has been retained by the broker. The broker sets this option to notify a subscriber that this publication was published earlier and has been retained, provided that the subscription has been registered with the InformIfRetained option. It is set only in response to a Register Subscriber or Request Update command message. Retained publications that are sent directly to subscribers do not have this option set.

Local
(*MQPSC_LOCAL*)

This option tells the broker that this publication should not be sent to other brokers. All subscribers that registered at this broker receive this publication if they have matching subscriptions.

OtherSubsOnly
(*MQPSC_OTHER_SUBS_ONLY*)

This option allows simpler processing of conference-type applications, where a publisher is also a subscriber to the same topic. It tells the broker not to send the publication to the publisher's subscriber queue even if it has a matching subscription. The publisher's subscriber queue consists of its QMgrName, QName, and optional CorrelId, as described below.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the MQMD (which must not be zero) is part of the publisher's subscriber queue, in applications where the publisher is also a subscriber.

None
(*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the publication options property. If other options are specified at the same time, None is ignored.

The default, if this property is omitted, is that no publication options are set.

<PubTime> (*MQPSC_PUBLISH_TIMESTAMP***)**
The value is an optional publication timestamp set by the publisher. It is 16 characters long with format:

```
YYYYMMDDHHMMSSTH
```

using Universal Time. This information is not checked by the broker before being sent to the subscribers.

<SeqNum> (*MQPSC_SEQUENCE_NUMBER***)**
The value is an optional sequence number set by the publisher.

It should be incremented by 1 with each publication. However, this is not checked by the broker, which merely transmits this information to subscribers.

If publications on the same topic are published to different interconnected brokers, it is the responsibility of the publishers to ensure that sequence numbers, if used, are meaningful.

<QMgrName> (*MQPSC_Q_MGR_NAME***)**
The value is a string containing the name of the queue manager for the publisher's subscriber queue, in applications where the publisher is also a subscriber (see OtherSubsOnly).

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the broker's queue manager.

<QName> (*MQPSC_Q_NAME***)**
The value is a string containing the name of the publisher's subscriber queue, in applications where the publisher is also a subscriber (see OtherSubsOnly).

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank if OtherSubsOnly is set.

Example

Here are some examples of *NameValueData* for a Publish command message.

The first example is for a publication sent by the match simulator in the sample application to indicate that a match has started.

```
<psc>
 <Command>Publish</Command>
 <Topic>Sport/Soccer/Event/MatchStarted</Topic>
</psc>
```

The second example is for a retained publication. The latest score in the match between Team1 and Team2 is published.

```
<psc>
 <Command>Publish</Command>
 <PubOpt>RetainPub</PubOpt>
 <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Register Subscriber message

The Register Subscriber command message is sent to a broker by a subscriber, or by another application on behalf of a subscriber, to indicate that it wants to subscribe to one or more topics at a subscription point. A message content filter can also be specified.

In publish/subscribe filter expressions, nesting parentheses causes performance to decrease exponentially. In practice, avoid nesting parentheses to a depth greater than about 6.

The message is sent to SYSTEM.BROKER.CONTROL.QUEUE, which is the broker's control queue. Authority to put a message to this queue is required, in addition to access authority (set by the broker's system administrator) for the topic, or topics, in the subscription.

If the user has authority on some, but not all, topics, only those with authority are registered; a warning response indicates those that are not registered.

See “Broker Response message” on page 88 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the broker.

If the queue is a temporary dynamic queue, the subscription is deregistered automatically by the broker when the queue is closed.

Properties

<Command> (*MQPSC_COMMAND***)**
The value is RegSub (*MQPSC_REGISTER_SUBSCRIBER*). This property must be specified.

<Topic> (*MQPSC_TOPIC***)**
The topic for which the subscriber wants to receive publications. Wildcard characters can be specified as part of the topic.

This property is required, and can optionally be repeated for as many topics as needed.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT***)**
The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

<Filter> (*MQPSC_FILTER***)**
The value is an SQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber.

If this property is omitted, no content filtering takes place.

<RegOpt> (*MQPSC_REGISTRATION_OPTION***)**
This Registration Options property can take the following values:

AddName
(*MQPSC_ADD_NAME*)

When specified for an existing subscription that matches the traditional identity of this Register Subscription command, but with no current **SubName** value, the **SubName** specified in this command is added to the subscription.

If **AddName** is specified the **SubName** field is mandatory, otherwise MQRCCF_REG_OPTIONS_ERROR is returned.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD) is used when sending matching publications to the subscriber queue. The CorrelId must not be zero,

FullResp
(*MQPSC_FULL_RESPONSE*)

When specified all attributes of the subscription are returned in the response message, if the command does not fail.

FullResp is valid only when the command message refers to a single subscription. Therefore, only one topic is permitted in the command; otherwise the command fails with return code *MQRCCF_REG_OPTIONS_ERROR*.

InformIfRet
(*MQPSC_INFORM_IF_RETAINED*)

The broker informs the subscriber if a publication is retained when it sends a Publish message in response to a Register Subscriber or Request Update command message. The broker does this by including the IsRetainedPub publication option in the message.

JoinExcl
(*MQPSC_JOIN_EXCLUSIVE*)

This option indicates that the specified **SubIdentity** should be added as the exclusive member of the identity set for the subscription, and that no other identities can be added to the set.

If the identity has already joined 'shared' and is the sole entry in the set, the set is changed to an exclusive lock held by this identity. Otherwise, if the subscription currently has other identities in the identity set (with shared access) the command fails with return code *MQRCCF_SUBSCRIPTION_IN_USE*.

JoinShared
(*MQPSC_JOIN_SHARED*)

This option indicates that the specified **SubIdentity** should be added to the identity set for the subscription.

If the subscription is currently locked exclusively (using the **JoinExcl** option), the command fails with return code *MQRCCF_SUBSCRIPTION_LOCKED*, unless the identity that has the subscription locked is the same identity as that in this command message. In this case the lock is automatically modified to a shared lock.

Local
(*MQPSC_LOCAL*)

The subscription is local and is not distributed to other brokers in the network. Publications made at other brokers are not delivered to this subscriber, unless it also has a corresponding global subscription.

NewPubsOnly
(*MQPSC_NEW_PUBS_ONLY*)

Retained publications that exist at the time the subscription is registered are not sent to the subscriber; only new publications are sent.

If a subscriber re-registers and changes this option so that it is no longer set, a publication that has already been sent to it might be sent again.

NoAlter
(*MQPSC_NO_ALTER*)

The attributes of an existing matching subscription is not changed.

When a subscription is being created, this option is ignored. All other options specified apply to the new subscription.

If a **SubIdentity** also has one of the join options (**JoinExcl** or **JoinShared**) specified, the identity is added to the identity set regardless of whether **NoAlter** is specified.

None
(*MQPSC_NONE*)

All registration options take their default values.

If the subscriber is already registered, its options are reset to their default values (note that this does *not* have the same affect as omitting the registration options property), and the subscription expiry is updated from the MQMD of the Register Subscriber message.

If other registration options are specified at the same time, None is ignored.

NonPers
(*MQPSC_NON_PERSISTENT*)

Publications matching this subscription are delivered to the subscriber as non-persistent messages.

Pers (*MQPSC_PERSISTENT*)

Publications matching this subscription are delivered to the subscriber as persistent messages.

PersAsPub
(*MQPSC_PERSISTENT_AS_PUBLISH*)

Publications matching this subscription are delivered to the subscriber with the persistence specified by the publisher. This is the default behavior.

PersAsQueue
(*MQPSC_PERSISTENT_AS_Q*)

Publications matching this subscription are delivered to the subscriber with the persistence specified on the subscriber queue.

PubOnReqOnly
(*MQPSC_PUB_ON_REQUEST_ONLY*)

The broker does not send publications to the subscriber, except in response to a Request Update command message.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the broker that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity *MQRCCF_DUPLICATE_SUBSCRIPTION* is returned.

This allows any user to modify or deregister the subscription if the user has suitable authority. There is therefore no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription of the request update command has **VariableUserId** set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the request update command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

If a Register Subscriber command message without this option set refers to an existing subscription which has this option set, the option is removed from this subscription and the userid of the subscription is now fixed. If there already exists a subscriber which has the same identity (queue, qmgr and correlid) but with a different userid associated to it, the command fails with return code *MQRCCF_DUPLICATE_IDENTITY* because there can only be one userid associated with a subscriber identity.

If the registration options property is omitted and the subscriber is already registered, its registration options are not changed and the subscription expiry is updated from the MQMD of the Register Subscriber message.

If the subscriber is not already registered, a new subscription is created with all registration options taking their default values.

The default values are PersAsPub and no other options set.

<QMgrName> (*MQPSC_Q_MGR_NAME***)**
The value is the name of the queue manager for the subscriber queue, to which matching publications are sent by the broker.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the broker's QMgrName.

<QName> (*MQPSC_Q_NAME***)**
The value is the name of the subscriber queue, to which matching publications are sent by the broker.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank in this case.

If the queue is a temporary dynamic queue, nonpersistent delivery of publications (NonPers) must be specified in the <RegOpt> property.

If the queue is a temporary dynamic queue, the subscription is deregistered automatically by the broker when the queue is closed.

<SubName> (*MQPSC_SUBSCRIPTION_NAME***)**
This is a name given to a particular subscription. You can use it instead of the queue manager, queue and optional correlId to refer to a subscription.

If a subscription already exists with this **SubName**, any other attributes of the subscription (Topic, QMgrName, QName, CorrelId, UserId, RegOpts, UserSubData, and Expiry) are overridden with the attributes, if specified, that are passed in the new Register Subscriber command message. However, if **SubName** is used with no QName field specified, and a ReplyToQ is specified in the MQMD header, the subscriber queue is changed to be the ReplyToQ.

If a subscription that matches the traditional identity of this command already exists, but has no **SubName**, the Registration command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*, unless the **AddName** option is specified.

If you try to alter an existing named subscription by using another Register Subscriber command that specifies the same **SubName**, and the values of Topic, QMgrName, QName, and CorrelId in the new command match a different existing subscription, with or without a SubName defined, the command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*. This prevents two subscription names referring to the same subscription.

<SubIdentity> (*MQPSC_SUBSCRIPTION_IDENTITY***)**
This string is used to represent an application with an interest in a subscription. It is a variable-length character string whose maximum length is 64 characters, and is optional. The broker maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to contain only one identity, or an unlimited number of identities (see the **JoinShared** and **JoinExcl** options).

A subscribe command that specifies the **JoinShared** or **JoinExcl** option adds the **SubIdentity** to the subscription's identity set, if it is not already there and if the existing set of identities allows such an action; that is, no other subscriber has joined exclusively or the identity set is empty.

Any alteration of the subscription's attributes as the result of a Register Subscriber command in which a **SubIdentity** is specified, only succeeds if it would be the only member of the set of identities for this subscription. Otherwise the command fails with return code *MQRCCF_SUBSCRIPTION_IN_USE*. This prevents a subscription's attributes from changing without other interested subscribers being aware.

If you specify a character string that is longer than 64 characters, the command fails with return code *MQRCCF_SUB_IDENTITY_ERROR*.

<SubUserData> (*MQPSC_SUBSCRIPTION_USER_DATA***)**
This is a variable-length text string. The value is stored by the broker with the subscription, but has no influence on publication delivery to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is there for the use of the application.

The **SubUserData** is returned in the Metatopic information (*MQCACF_REG_SUB_USER_DATA*) for a subscription if present.

If you specify more than one of the registration option values `NonPers`, `PersAsPub`, `PersAsQueue`, and `Pers`, then only the last one is used. You cannot combine these options in an individual subscription.

Example

Here is an example of NameValueData for a Register Subscriber command message. In the sample application, the results service uses this message to register a subscription to the topics containing the latest scores in all matches, with the 'Publish on Request Only' option set. The subscriber's identity, including the CorrelId, is taken from the defaults in the MQMD.

```
<psc>
 <Command>RegSub</Command>
 <RegOpt>PubOnReqOnly</RegOpt>
 <RegOpt>CorrelAsId</RegOpt>
 <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Request Update message

The Request Update command message is sent from a subscriber to a broker, to request the current retained publications for the specified topic and subscription point that match the given (optional) filter.

This message is sent to *SYSTEM.BROKER.CONTROL.QUEUE*, the broker's control queue. Authority to put a message to this queue is required, in addition to access authority for the topic in the request update; this is set by the broker's system administrator.

This command is normally used if the subscriber specified the option PubOnReqOnly when it registered. If the broker has any matching retained publications, they are sent to the subscriber. If the broker has no matching retained publications, the request fails with return code *MQRCCF_NO_RETAINED_MSG*. The requester must have previously registered a subscription with the same Topic, SubPoint, and Filter values.

Properties

<Command> (*MQPSC_COMMAND***)**
The value is ReqUpdate (*MQPSC_REQUEST_UPDATE*). This property must be specified.

<Topic> (*MQPSC_TOPIC***)**
The value is the topic that the subscriber is requesting; wildcard characters are allowed.

This property must be specified, but only one occurrence is allowed in this message.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT***)**
The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

<Filter> (*MQPSC_FILTER***)**
The value is an ESQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber.

The <Filter> property should have the same value as that specified on the original subscription for which you are now requesting an update.

If this property is omitted, no content filtering takes place.

<RegOpt> (*MQPSC_REGISTRATION_OPTION***)**
The registration options property can take the following value:

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD), which must not be zero, is used when sending matching publications to the subscriber queue.

None
(*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the <RegOpt> property. If other options are specified at the same time, None is ignored.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager, and correlid) is not restricted to a single userid. This differs from the existing behavior of the broker that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*.

This allows any user to modify or deregister the subscription when they have suitable authority. Therefore, there is no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription, the command must come from the same userid as the original subscription.

If the subscription of the Request Update command has **VariableUserId** set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the Request Update command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

<QMgrName> (*MQPSC_Q_MGR_NAME***)**
The value is the name of the queue manager for the subscriber queue, to which the matching retained publication is sent by the broker.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the broker's QMgrName.

<QName> (*MQPSC_Q_NAME***)**
The value is the name of the subscriber queue, to which the matching retained publication is sent by the broker.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank in this case.

<SubName> (*MQPSC_SUBSCRIPTION_NAME***)**
This is a name given to a particular subscription. If specified on a Request Update command the **SubName** value takes precedence over all other identifier fields except the userid, unless VariableUserId is set on the subscription itself. If **VariableUserId** is not set, the Request Update command succeeds only if the userid of the command message matches that of the subscription. If the userid of the command message does not match that of the subscription, the command fails with return code *MQRCCF_DUPLICATE_IDENTITY*.

If **VariableUserId** is set, and the userid differs from that of the subscription, the command succeeds if the userid of the new command message has authority to browse the stream queue and put to the subscriber queue of the subscription. Otherwise, the command fails with return code *MQRCCF_NOT_AUTHORIZED*.

If a subscription exists that matches the traditional identity of this command, but has no **SubName**, theRequest Update command fails with return code *MQRCCF_SUB_NAME_ERROR*.

If an attempt is made to request an update for a subscription that has a **SubName** using a command message that matches the traditional identity, but with no **SubName** specified, the command succeeds.

Example

Here is an example of NameValueData for a Request Update command message. In the sample application, the results service uses this message to request retained publications containing the latest scores for all teams. The subscriber's identity, including the CorrelId, is taken from the defaults in the MQMD.

```
<psc>
 <Command>ReqUpdate</Command>
 <RegOpt>CorrelAsId</RegOpt>
 <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Broker Response message

A Broker Response message is sent from a broker to the ReplyToQ of a publisher or a subscriber, to indicate the success or failure of a command message received by the broker if the command message descriptor specified that a response is required.

The response message is contained within the NameValueData field of the MQRFH2 header, in a `<pscr>` folder.

In the case of a warning or error, the response message contains the `<psc>` folder from the command message as well as the `<pscr>` folder. The message data, if any, is not contained in the broker response message. In the case of an error, none of the message that caused an error has been processed; in the case of a warning, some of the message might have been processed successfully.

If there is a failure sending a response:

- For publication messages, the broker tries to send the response to the WebSphere MQ dead-letter queue if the MQPUT fails. This allows the publication to be sent to subscribers even if the response cannot be sent back to the publisher.
- For other messages, or if the publication response cannot be sent to the dead-letter queue, an error is logged and the command message is normally rolled back. Whether this happens depends on how the MQInput node has been configured.

Properties

<Completion> (*MQPSCR_COMPLETION***)**
The completion code, which can take one of three values:
ok Command completed successfully
warning
Command completed but with warning
error Command failed

<Response> (*MQPSCR_RESPONSE***)**
The response to a command message, if that command produced a completion code of warning or error. It contains a <Reason> property, and might contain other properties that indicate the cause of the warning or error.

In the case of one or more errors, there is only one response folder, indicating the cause of the first error only. In the case of one or more warnings, there is a response folder for each warning.

<Reason> (*MQPSCR_REASON***)**
The reason code qualifying the completion code, if the completion code is a warning or error. It is set to one of the error codes listed below. The <Reason> property is contained within a <Response> folder. The reason code can be followed by any valid property from the <psc> folder (for example, a topic name), indicating the cause of the error or warning.

Examples

Here are some examples of NameValueData in a Broker Response message. A successful response might be the following:

```
<pscr>
  <Completion>ok</Completion>
</pscr>
```

Here is an example of a failure response; the failure is a filter error. The first NameValueData string contains the response; the second contains the original command.

```
<pscr>
  <Completion>error</Completion>
  <Response>
    <Reason>3150</Reason>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the broker is responding)
 ...
</psc>
```

Here is an example of a warning response (due to unauthorized topics). The first NameValueData string contains the response; the second NameValueData string contains the original command.

```
<pscr>
  <Completion>warning</Completion>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic1</Topic>
  </Reponse>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic2</Topic>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the broker is responding)
 ...
</psc>
```

Reason codes

The following reason codes might be returned in the Reason field of a publish/subscribe response <pscr> folder. Constants that can be used to represent these codes in the C or C++ programming languages are also given. The MQRC_ constants require the WebSphere MQ cmqc.h header file. The MQRCCF_ constants require the WebSphere MQ cmqcfc.h header file (apart from

MQRCCF_FILTER_ERROR and *MQRCCF_WRONG_USER*, which require the WebSphere Event Broker BipRfc.h header file).

Reason code and text	Explanation	Issued by
2336 MQRC_RFH_COMMAND_ERROR	Valid values for the <Command> field of a <psc> folder are: RegSub, DeregSub, Publish, DeletePub, and ReqUpdate. Any other values result in this error code being issued.	Any command
2337 MQRC_RFH_ PARM_ERROR	The <psc> and <mcd> folders both have a set of valid parameters that can be specified within them. Check the descriptions of these folders and ensure that you have not specified incorrect parameters.	Any command
2338 MQRC_RFH_DUPLICATE_PARM	Some parameters (for example, Topic) within a <psc> folder can be repeated, but others (for example, Command) cannot be repeated. Check that you have not duplicated a non-repeatable parameter.	Any command
2339 MQRC_RFH_PARM_MISSING	Some parameters within <psc> or <mcd> folders are optional and can be omitted; some are mandatory and must not be omitted. Check that you have included all mandatory parameters within your <psc> and <mcd> folders.	Any command
3008 MQRCCF_COMMAND_FAILED	An internal error occurred which prevented the command from executing correctly. The error might occur if the command is reissued. The system event log for the broker contains information which should be used when reporting the problem to IBM.	Any command
3072 MQRCCF_TOPIC_ERROR	One or more of the values you supplied for the Topic parameter are incorrect. Check that your values for Topic conform to the specified restrictions.	Any command
3073 MQRCCF_NOT_REGISTERED	The combination of SubPoint, Topic, and Filter that you specified on your DeregSub or ReqUpdate command was either not a combination with which you had previously registered or, for the DeregSub command if the DeregAll option was specified, one of the SubPoint, Topic, or Filter properties was not used to deregister any subscription.	Deregister Subscriber and Request Update commands
3074 MQRCCF_Q_MGR_NAME_ERROR	The specified queue manager was not valid, or the queue manager was not available or did not exist.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3076 MQRCCF_Q_NAME_ERROR	The specified queue name was not valid, or the queue did not exist on the specified queue manager.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands

Reason code and text	Explanation	Issued by
3077 MQRCCF_NO_RETAINED_MSG	There were no retained messages for the topic you specified. This might or might not be an error, depending on the design of your application program.	Request Update command
3079 MQRCCF_INCORRECT_Q	RegSub, DeregSub, and ReqUpdate commands are always sent to the SYSTEM.BROKER.CONTROL.QUEUE queue of the broker for which they are intended. Publish and Delete Publication commands are sent to the input queue for the particular publish/subscribe message flow for which they are intended; this is determined when the message flow is designed. This error code is returned if a command is sent to the wrong queue.	Any command
3080 MQRCCF_CORREL_ID_ERROR	You have specified CorrelAsId as one of your RegOpt parameters. However, the CorrelId field of the MQMD does not contain a valid correlation identifier (that is, it is set to MQCI_NONE).	Deregister Subscriber and Register Subscriber commands
3081 MQRCCF_NOT_AUTHORIZED	You are not authorized to perform the requested action. Authorization settings for the broker are handled by the system administrator using the Topics Hierarchy editor.	Publish and Register Subscriber commands
3083 MQRCCF_REG_OPTIONS_ERROR	You have specified an unrecognized RegOpt parameter in the <psc> folder that contains your RegSub or DeregSub command.	Deregister Subscriber and Register Subscriber commands
3084 MQRCCF_PUB_OPTIONS_ERROR	You have specified an unrecognized PubOpt parameter in the <psc> folder that contains your Publish command.	Publish command
3087 MQRCCF_DEL_OPTIONS_ERROR	You have specified an unrecognized DelOpt parameter in the <psc> folder that contains your DeletePub command.	Delete Publication command
3150 MQRCCF_FILTER_ERROR	The value specified for the Filter parameter is not valid. Check the section that describes the valid syntax for filter expressions and ensure that your expression conforms.	Deregister Subscriber, Register Subscriber, and Request Update commands
3151 MQRCCF_WRONG_USER	A subscription that matches the one specified already exists; however, it was registered by a different user. A subscription can only be changed or deregistered by the user who originally registered it.	Deregister Subscriber, Register Subscriber, and Request Update commands
3152 MQRCCF_DUPLICATE_SUBSCRIPTION	A matching subscription already exists with a different subscription name.	
3153 MQRCCF_SUB_NAME_ERROR	Either the format of the subscription name is not valid, or a matching subscription already exists with no subscription name.	

Reason code and text	Explanation	Issued by
3154 MQRCCF_SUB_IDENTITY_ERROR	The subscription identity parameter is in error. Either the supplied value exceeds the maximum length allowed, or the subscription identity is not currently a member of the subscription's identity set and a Join registration option was not specified.	
3155 MQRCCF_SUBSCRIPTION_IN_USE	An attempt to modify or deregister a subscription was attempted by a member of the identity set when it was not the only member of this set.	
3156 MQRCCF_SUBSCRIPTION_LOCKED	The subscription is currently exclusively locked by another identity.	
3157 MQRCCF_ALREADY_JOINED	A Join registration option was specified but the subscriber identity was already a member of the subscription's identity set.	

MQMD settings in command messages to the broker

Applications that send command messages to the broker use the following settings of fields in the message descriptor (MQMD). Fields that are left as the default value, or can be set to any valid value in the usual way, are not listed here.

Report
See MsgType and CorrelId (below).

MsgType
MsgType should be set to *MQMT_REQUEST* for a command message if a response is always required. The MQRO_PAN and MQRO_NAN flags in the Report field are not significant in this case.

If MsgType is set to MQMT_DATAGRAM, responses depend on the setting of the MQRO_PAN and MQRO_NAN flags in the Report field:

- MQRO_PAN alone means that the broker sends a response only if the command succeeds.
- MQRO_NAN alone means that the broker sends a response only if the command fails.
- If a command completes with a warning, a response is sent if either MQRO_PAN or MQRO_NAN is set.
- MQRO_PAN + MQRO_NAN means that the broker sends a response whether the command succeeds or fails. This has the same effect from the broker's perspective as setting MsgType to MQMT_REQUEST.
- If neither MQRO_PAN nor MQRO_NAN is set, no response is ever sent.

Format
Set to MQFMT_RF_HEADER_2

MsgId
This field is normally set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
This field can be set to any value. If the sender's identity includes a CorrelId,

specify this value, together with MQRO_PASS_CORREL_ID in the Report field, to ensure that it is set in all response messages sent by the broker to the sender.

ReplyToQ
This field defines the queue to which responses, if any, are to be sent. This might be the sender's queue; this has the advantage that the QName parameter can be omitted from the message. If, however, responses are to be sent to a different queue, the QName parameter is needed.

ReplyToQMgr
This field defines the queue manager for responses. If you leave this field blank (the default value), the local queue manager puts its own name in this field.

MQMD settings for publications forwarded by a broker

A broker uses the following settings of fields in the message descriptor (MQMD) when it sends a publication to a subscriber. All other fields in the MQMD are set to their default values.

Report
Report is set to MQRO_NONE.

MsgType
MsgType is set to MQMT_DATAGRAM.

Expiry
Expiry is set to the value in the Publish message received from the publisher. In the case of a retained message, the time outstanding is reduced by the approximate time that the message has been at the broker.

Format
Format is set to MQFMT_RF_HEADER_2

MsgId
MsgId is set to a unique value.

CorrelId
If CorrelId is part of the subscriber's identity, this is the value specified by the subscriber when registering. Otherwise, it is a non-zero value chosen by the broker.

Priority
Priority takes the value set by the publisher, or as resolved if the publisher specified MQPRI_PRIORITY_AS_Q_DEF.

Persistence
Persistence takes the value set by the publisher, or as resolved if the publisher specified MQPER_PERSISTENCE_AS_Q_DEF, unless specified otherwise in the Register Subscriber message for the subscriber to which this publication is being sent.

ReplyToQ
ReplyToQ is set to blanks.

ReplyToQMgr
ReplyToQMgr is set to the name of the broker's queue manager.

UserIdentifier
UserIdentifier is the subscriber's user identifier, as set when the subscriber registered.

AccountingToken
AccountingToken is the subscriber's accounting token, as set when the subscriber first registered.

ApplIdentityData
ApplIdentityData is the subscriber's application identity data, as set when the subscriber first registered.

PutApplType
PutApplType is set to MQAT_BROKER.

PutApplName
PutApplName is set to the first 28 characters of the name of the broker's queue manager.

PutDate
PutDate is the timestamp when the broker puts the message.

PutTime
PutTime is the timestamp when the broker puts the message.

ApplOriginData
ApplOriginData is set to blanks.

MQMD settings in broker response messages

A broker uses the following settings of fields in the message descriptor (MQMD) when sending a reply to a publication message. All other fields in the MQMD are set to their default values.

Report
Report is set to all zeroes.

MsgType
MsgType is set to MQMT_REPLY.

Format
Format is set to MQFMT_RF_HEADER_2

MsgId
The setting of MsgId depends on the Report options in the original command message. By default, it is set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
The setting of CorrelId depends on the Report options in the original command message. By default, this means that the CorrelId is set to the same value as the MsgId of the command message. This can be used to correlate commands with their responses.

Priority
Priorityis set to the same value as in the original command message.

Persistence
Persistence is set to the value set in the original command message.

Expiry
Expiry is set to the same value as in the original command message received by the broker.

PutApplType
PutApplType is set to MQAT_BROKER.

PutApplName
PutApplName is set to the first 28 characters of name of the queue manager.

Other context fields are set as if generated with MQPMO_PASS_IDENTITY_CONTEXT.

Part 4. Appendixes

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032,
Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not

been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	CICS	Cloudscape
DB2	DB2 Connect	DB2 Universal Database
developerWorks	Domino	
Everyplace	FFST	First Failure Support Technology
IBM	IBMLink	IMS
IMS/ESA	iSeries	Language Environment
Lotus	MQSeries	MVS
NetView	OS/400	OS/390
pSeries	RACF	Rational
Redbooks	RETAIN	RS/6000
SupportPac	Tivoli	VisualAge
WebSphere	xSeries	z/OS
zSeries		

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

Z

Printed in USA